Cat. No. I55E-EN-01

Programmable Controller
SYSMAC CJ-series
CJ1W-MCH72

Motion Control Unit

trajexia

OPERATION MANUAL

OMRON

Notice

OMRON products are manufactured for use by a trained operator and only for the purposes described in this manual.

The following conventions are used to classify and explain the precautions in this manual. Always heed the information provided with them.

⚠ **WARNING** Indicates information that, if not heeded, could possibly result in serious injury or loss of life.

⚠ **Caution** Indicates information that, if not heeded, could possibly result in minor or relatively serious injury, damage to the product or faulty operation.

OMRON product references

All OMRON products are capitalized in this manual.
The first letter of the word *Unit* is also capitalized when it refers to an OMRON product, regardless of whether it appears in the proper name of the product.
The abbreviation PLC means Programmable Logic Controller.

Visual aids

The following headings appear in the left column of the manual to help you locate different types of information.

Note Indicates information of particular interest for efficient and convenient operation of the product.

Trademarks and copyrights

MECHATROLINK is a registered trademark of Yaskawa Corporation.
Trajexia is a registered trademark of OMRON.
All other product names, company names, logos or other designations mentioned herein are trademarks of their respective owners.

Copyright

Copyright © 2008 OMRON
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, mechanical, electronic, photocopying, recording, or otherwise, without the prior written permission of OMRON.
No patent liability is assumed with respect to the use of the information contained herein. Moreover, because OMRON is constantly striving to improve its high-quality products, the information contained in this manual is subject to change without notice. Every precaution has been taken in the preparation of this manual. Nevertheless, OMRON assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained in this publication.

TABLE OF CONTENTS

Precautions vii

1 Intended audience vii
2 General precautions vii
3 Safety precautions vii
4 Operating environment precautions viii
5 Application precautions ix
6 Conformance to EC Directives xi

SECTION 1
Introduction 1

1-1 Overview 1
1-2 System philosophy 2
1-3 Motion control concepts 3
1-4 Servo system principles 12
1-5 Trajexia system architecture 15
1-6 Cycle time 16
1-7 Program control and multi-tasking 20
1-8 Motion sequence and axes 22
1-9 Motion buffers 29
1-10 Mechanical system 30
1-11 Axis numbers 31

SECTION 2
Installation and wiring 33

2-1 Unit components 33
2-2 Wiring 38
2-3 Installation 48
2-4 Specifications 52

SECTION 3
Data exchange 55

3-1 Introduction 55
3-2 Memory areas 55
3-3 Data 56
3-4 FINS commands 58

SECTION 4
BASIC commands 65

4-1 Categories 65
4-2 All BASIC commands 75

SECTION 5
Examples 243

5-1 How-to’s 243
5-2 Practical examples 293

TABLE OF CONTENTS

SECTION 6
Troubleshooting 315

6-1 System errors 315
6-2 Data exchange errors 315

Revision history 323

Precautions

1 Intended audience

This manual is intended for the following personnel, who must also have knowledge of electrical systems (an electrical engineer or the equivalent).

- Personnel in charge of installing FA systems.
- Personnel in charge of designing FA systems.
- Personnel in charge of managing FA systems and facilities.

2 General precautions

The user must operate the product according to the performance specifications described in the operation manuals.

Before using the product under conditions which are not described in the manual or applying the product to nuclear control systems, railroad systems, aviation systems, vehicles, combustion systems, medical equipment, amusement machines, safety equipment, and other systems, machines, and equipment that may have a serious influence on lives and property if used improperly, consult your OMRON representative.

Make sure that the ratings and performance characteristics of the product are sufficient for the systems, machines, and equipment, and be sure to provide the systems, machines, and equipment with double safety mechanisms.

This manual provides information for using the CJ1W-MCH72. Be sure to read this manual before attempting to use the Unit and keep this manual close at hand for reference during operation.

⚠WARNING It is extremely important that the CJ1W-MCH72 and related devices be used for the specified purpose and under the specified conditions, especially in applications that can directly or indirectly affect human life. You must consult with your OMRON representative before applying the CJ1W-MCH72 and related devices to the above mentioned applications.

3 Safety precautions

⚠WARNING Never short-circuit the positive and negative terminals of the batteries, charge the batteries, disassemble them, deform them by applying pressure, or throw them into a fire.
The batteries may explode, combust or leak liquid.

⚠WARNING The CJ1W-MCH72 outputs will go off due to overload of the output transistors (protection). As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.

⚠WARNING The CJ1W-MCH72 will turn off the WDOG when its self-diagnosis function detects any error. As a countermeasure for such errors, external safety measures must be provided to ensure safety in the system.

⚠WARNING Never attempt to disassemble any Units while power is being supplied. Doing so may result in serious electric shock.

⚠WARNING Do not attempt to disassemble, repair, or modify any Units. Any attempt to do so may result in malfunction, fire, or electric shock.

⚠WARNING Never touch any of the terminals while power is being supplied. Doing so may result in serious electric shock.

⚠WARNING Provide safety measures in external circuits (i.e., not in the Programmable Controller) to ensure safety in the system if an abnormality occurs due to malfunction of the PLC, malfunction of the CJ1W-MCH72, or external factors affecting the operation of the PLC or CJ1W-MCH72. Not providing sufficient safety measures may result in serious accidents.

- Emergency stop circuits, interlock circuits, limit circuits, and similar safety measures must be provided in external control circuits.
- The PLC will turn OFF all outputs when its self-diagnosis function detects any error or when a severe failure alarm (FALS) instruction is executed. As a countermeasure for such errors, external safety measures must be provided to ensure safety in the system.
- The PLC or CJ1W-MCH72 outputs may remain ON or OFF due to deposits on or burning of the output relays, or destruction of the output transistors. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.
- When the 24 V DC output (service power supply to the PLC) is overloaded or short-circuited, the voltage may drop and result in the outputs being turned OFF. As a countermeasure for such problems, external safety measures must be provided to ensure safety in the system.
- External safety measures must also be taken to ensure safety in the event of unexpected operation when connecting or disconnecting the connectors of the CJ1W-MCH72.

⚠Caution User programs written to the CJ1W-MCH72 will not be automatically backed up in the CJ1W-MCH72 flash memory (flash memory function).

⚠Caution Tighten the screws on the terminal block of the Power Supply Unit to the torque specified in this manual. Loose screws may result in burning or malfunction.

⚠Caution When positioning to a position determined using the teaching function, set the position designation setting in the positioning sequence to absolute positioning. If it is set to relative positioning, positioning will be performed to a position other than the one obtained with the teaching function.

⚠Caution Execute online edit only after confirming that no adverse effects will be caused by extending the cycle time. Otherwise, the input signals may not be readable.

⚠Caution Confirm the safety of the destination node before transferring a program to the node or changing the contents of I/O memory. Doing either of these without confirming safety may result in injury.

⚠Caution Do not save data into the flash memory during memory operation or while the motor is running. Otherwise, unexpected operation may be caused.

4 Operating environment precautions

⚠Caution Do not operate the control system in the following locations:

- Locations subject to direct sunlight.
- Locations subject to temperatures or humidity outside the range specified in the specifications.
- Locations subject to condensation as the result of severe changes in temperature.
- Locations subject to corrosive or flammable gases.
- Locations subject to dust (especially iron dust) or salts.
- Locations subject to exposure to water, oil, or chemicals.
- Locations subject to shock or vibration.

⚠Caution Take appropriate and sufficient countermeasures when installing systems in the following locations:

- Locations subject to static electricity or other forms of noise.
- Locations subject to strong electromagnetic fields.

- Locations subject to possible exposure to radioactivity.
- Locations close to power supplies.

⚠Caution The operating environment of the PLC System can have a large effect on the longevity and reliability of the system. Improper operating environments can lead to malfunction, failure, and other unforeseeable problems with the PLC System. Be sure that the operating environment is within the specified conditions at installation and remains within the specified conditions during the life of the system.

5 Application precautions

⚠WARNING Do not start the system until you check that the axes are present and of the correct type. The numbers of the axis will change if MECHATROLINK-II network errors occur during start-up or if the MECHATROLINK-II network configuration changes.

⚠WARNING Check the user program for proper execution before actually running it in the Unit. Not checking the program may result in an unexpected operation.

⚠WARNING Observe the following precautions when using the CJ1W-MCH72 or the PLC. Failure to abide by the following precautions could lead to serious or possibly fatal injury. Always heed these precautions.

- Always connect to a ground of 100 Ω or less when installing the Units. Not connecting to a ground of 100 Ω or less may result in electric shock.
- Always turn OFF the power supply to the PLC before attempting any of the following. Not turning OFF the power supply may result in malfunction or electric shock.
  - Mounting or dismounting Power Supply Units, I/O Units, CPU Units, Memory Cassettes, or any other Units.
  - Assembling the Units.
  - Setting DIP switches or rotary switches.
  - Connecting cables or wiring the system.
  - Connecting or disconnecting the connectors.

⚠Caution Be sure that all mounting screws, terminal screws, and cable connector screws are tightened to the torque specified in this manual. Incorrect tightening torque may result in malfunction.

⚠Caution Wire correctly. Incorrect wiring may result in burning.

⚠Caution Mount the Unit only after checking the terminal block completely.

⚠Caution Resume operation only after transferring to the new CJ1W-MCH72 Unit the contents of the VR and table memory required for operation. Not doing so may result in an unexpected operation.

⚠Caution When replacing parts, be sure to confirm that the rating of a new part is correct. Not doing so may result in malfunction or burning.

⚠Caution Use the dedicated connecting cables specified in operation manuals to connect the Units. Using commercially available RS-232C computer cables may cause failures in external devices or the Unit.

⚠Caution Outputs may remain on due to a malfunction in the built-in transistor outputs or other internal circuits. As a countermeasure for such problems, external safety measures must be provided to ensure the safety of the system.

⚠Caution Failure to abide by the following precautions may lead to faulty operation of the PLC, the CJ1W-MCH72 or the system, or could damage the PLC or CJ1W-MCH72. Always heed these precautions.

- Fail-safe measures must be taken by the customer to ensure safety in the event of incorrect, missing, or abnormal signals caused by broken signal lines, momentary power interruptions, or other causes.
- Interlock circuits, limit circuits, and similar safety measures in external circuits (i.e., not in the Programmable Controller) must be provided by the customer.
- Install external breakers and take other safety measures against short-circuiting in external wiring. Insufficient safety measures against short-circuiting may result in burning.
- Install the PLC Unit as far as possible from sources of strong harmonic noise.
- Lock the sliders securely until they click into place when connecting the Power Supply Unit, CPU Unit, I/O Units, Special I/O Units, or CPU Bus Units. Functions may not work correctly if the sliders are not locked properly.
- Always attach the End Cover provided with the CPU Unit to the Unit on the right end of the PLC. The CJ-series PLC will not operate properly if the End Cover is not attached.
- Always use the power supply voltages specified in the operation manuals. An incorrect voltage may result in malfunction or burning.
- Take appropriate measures to ensure that the specified power with the rated voltage and frequency is supplied in places where the power supply is unstable. An incorrect power supply may result in malfunction.
- Use crimp terminals for wiring. Do not connect bare stranded wires directly to terminals. Connection of bare stranded wires may result in burning.
- Leave the label attached to the Unit when wiring. Removing the label may result in malfunction if foreign matter enters the Unit.
- Remove the label after the completion of wiring to ensure proper heat dissipation. Leaving the label attached may result in malfunction.
- Do not apply voltages to the Input Units in excess of the rated input voltage. Excess voltages may result in burning.
- Do not apply voltages or connect loads to the Output Units in excess of the maximum switching capacity. Excess voltage or loads may result in burning.
- Check the user program for proper execution before actually running it on the Unit. Not checking the program may result in an unexpected operation.
- Be sure that the terminal blocks, Memory Units, expansion cables, and other items with locking devices are properly locked into place. Improper locking may result in malfunction.
- Double-check all wiring and switch settings before turning ON the power supply. Incorrect wiring may result in burning.
- Disconnect the LR and GR terminals when performing insulation resistance or withstand voltage tests. Not disconnecting the functional ground terminal may result in burning.

⚠Caution Confirm that no adverse effect will occur in the system before attempting any of the following. Not doing so may result in an unexpected operation.

- Changing the operating mode of the PLC (including the operating mode at power up).
- Force-setting/force-resetting any bit in memory.
- Changing the present value of any word or any set value in memory.

⚠Caution Do not pull on the cables or bend the cables beyond their natural limit. Doing either of these may break the cables.

⚠Caution Do not place objects on top of the cables or other wiring lines. Doing so may break the cables.

⚠Caution Resume operation only after transferring the system parameter data to the CJ1W-MCH72 and saving the data to flash memory. Not doing so may result in an unexpected operation.

⚠Caution Confirm that set parameters and data operate properly.

⚠Caution Check the pin numbers before wiring the connectors.

⚠Caution Perform wiring according to specified procedures.

⚠Caution Before touching a Unit, be sure to first touch a grounded metallic object in order to discharge any static build-up. Not doing so may result in malfunction or damage.

⚠Caution Do not drop the Unit or subject it to abnormal shock or vibration.

⚠Caution Confirm the safety of the destination node before transferring a program to the node or changing the contents of I/O memory. Doing either of these without confirming safety may result in injury.

6 Conformance to EC Directives

6-1 Applicable directives

- EMC Directives

6-2 Concepts

OMRON devices that comply with EC Directives also conform to the related EMC standards so that they can be more easily built into other devices or machines. The actual products have been checked for conformity to EMC standards (see the following note). Whether the products conform to the standards in the system used by the customer, however, must be checked by the customer.

EMC-related performance of the OMRON devices that comply with EC Directives will vary depending on the configuration, wiring, and other conditions of the equipment or control panel in which the OMRON devices are installed. The customer must, therefore, perform final checks to confirm that devices and the overall machine conform to EMC standards.

Note Applicable EMC (Electromagnetic Compatibility) standards are as follows:

- EMS (Electromagnetic Susceptibility): EN61000-6-2
- EMI (Electromagnetic Interference): EN61000-6-4
  (Radiated emission: 10-m regulations)

6-3 Conformance to EC Directives

The CJ1W-MCH72 complies with EC Directives. To ensure that the machine or device in which a CJ1W-MCH72 is used complies with EC Directives, the CJ1W-MCH72 must be installed as follows:

1 The CJ1W-MCH72 must be installed within a control panel.
2 Reinforced insulation or double insulation must be used for the DC power supplies used for the communications and I/O power supplies.
3 Units complying with EC Directives also conform to the Common Emission Standard (EN61000-6-4). With regard to the radiated emission (10-m regulations), countermeasures will vary depending on the devices connected to the control panel, wiring, the configuration of the system, and other conditions. The customer must, therefore, perform final checks to confirm that devices and the overall machine conform to EC Directives.

6-4 Installation within Control Panel

Unnecessary clearance in cable inlet or outlet ports, operation panel mounting holes, or in the control panel door may cause electromagnetic wave leakage or interference. In this case, the product may fail to meet EC Directives. In order to prevent such interference, fill clearances in the control panel with

conductive packing. (In places where conductive packing comes in contact with the control panel, ensure electrical conductivity by removing the paint coating or masking these parts when painting.)

SECTION 1
Introduction

1-1 Overview

The CJ1W-MCH72 is a Trajexia-style motion control unit that can be connected to a CJ1-series PLC. It acts as an interface between PLC systems and Trajexia-style motion control systems.

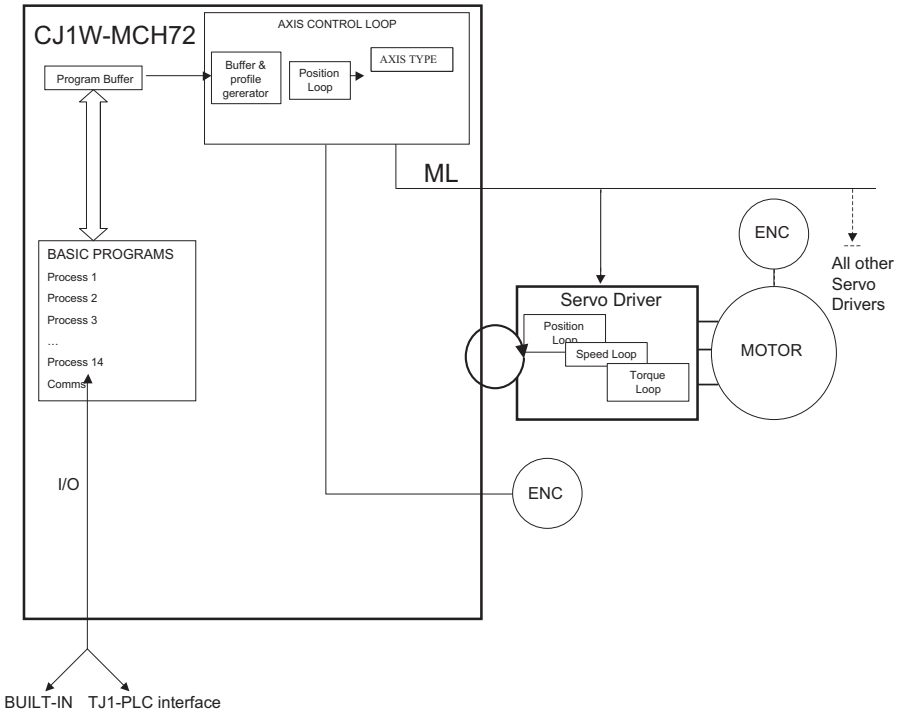
Trajexia is the OMRON motion platform that offers you the performance and the ease of use of a dedicated motion system. It maximum flexibility and scalability. At the heart of Trajexia lies the TJ1 multi-tasking motion coordinator. Powered by a 32-bit DSP, it can do motion tasks such as e-cam, e-gearbox, registration control and interpolation, all using simple motion commands.

The CJ1W-MCH72 has the following features:

- A MECHATROLINK-II connection for a MECHATROLINK-II network with up to 30 axes. The motion cycle time is selectable: 0.5 ms, 1 ms, 2 ms or 4 ms.
- An Encoder Interface connection. It supports the main absolute encoder protocols allowing the connection of an external encoder to the system.
- The possibility to exchange analogue and digital input and output data with the PLC CPU.
- A wide choice of rotary, linear and direct-drive servos as well as Inverters are available to fit your needs in compactness, performance and reliability. The Inverters connected to the MECHATROLINK-II are driven at the same update cycle time as the Servo Drivers.

Note The Trajexia system supports 3 kinds of MECHATROLINK-II slaves: Servo Drivers, Inverters and I/Os.
The CJ1W-MCH72 only supports 2 kinds of MECHATROLINK-II slaves: Servo Drivers and Inverters. It does not support I/Os.

1-2 System philosophy



The system philosophy is centred around the relationship between:
- System architecture
- Cycle time
- Program control and multi-tasking
- Motion sequence and axes
- Motion buffers

A clear understanding of the relationship between these concepts is necessary to obtain the best results for the Trajexia system.

1-2-1 Glossary

1-2-1-1 Motion sequence

The Motion Sequence is responsible for controlling the position of the axes.

1-2-1-2 Servo period

Defines the frequency at which the Motion Sequence is executed. The servo period must be set according to the configuration of the physical axes. The available settings are 0.5 ms, 1 ms, 2 ms or 4 ms.

1-2-1-3 Cycle time

Is the time needed to execute one complete cycle of operations in the CJ1W-MCH72. The cycle time is divided in 4 time slices of equal time length, called "CPU Tasks". The cycle time is 1ms if **SERVO_PERIOD** = 0.5 ms or **SERVO_PERIOD** = 1 ms, 2 ms if the **SERVO_PERIOD** = 2 ms and 4 ms if the **SERVO_PERIOD** = 4 ms

1-2-1-4 CPU tasks

The operations executed in each CPU task are:

CPU task	Operation
First CPU task	Motion Sequence Low priority process
Second CPU task	High priority process
Third CPU task	Motion Sequence (only if **SERVO_PERIOD** = 0.5 ms) LED Update High priority process
Fourth CPU task	External Communications

1-2-1-5 Program

A program is a piece of BASIC code.

1-2-1-6 Process

Is a program in execution with a certain priority assigned. Process 0 to 12 are Low priority processes and Process 13 and 14 are High priority processes. First the process priority, High or Low, and then the process number, from high to low, will define to which CPU task the process will be assigned.

1-3 Motion control concepts

The CJ1W-MCH72 offers these types of positioning control operations:

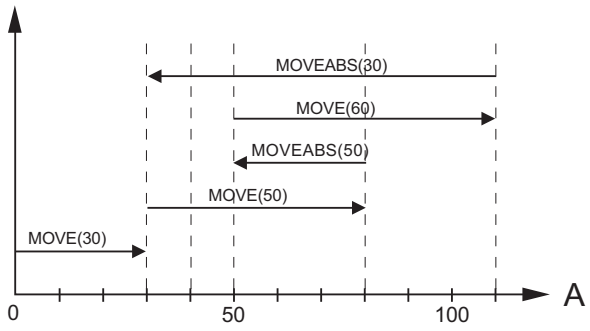
1 Point-to-Point (PTP) control
2 Continuous Path (CP) control
3 Electronic Gearing (EG) control.

This section introduces some of the commands and parameters used in the BASIC programming of the motion control application.

Coordinate system

Positioning operations performed by the CJ1W-MCH72 are based on an axis coordinate system. The CJ1W-MCH72 converts the position data from either the connected Servo Driver or the connected encoder into an internal absolute coordinate system.

The engineering unit that specifies the distances of travelling can be freely defined for each axis separately. The conversion is performed through the use of the unit conversion factor, which is defined by the **UNITS** axis parameter. The origin point of the coordinate system can be determined using the **DEFPOS** command. This command re-defines the current position to zero or any other value.



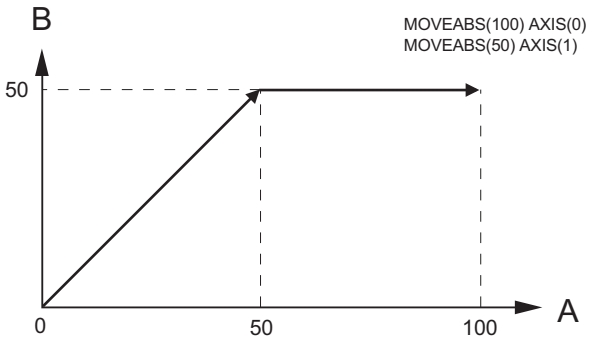
A move is defined in either absolute or relative terms. An absolute move takes the axis (A) to a specific predefined position with respect to the origin point. A relative move takes the axis from the current position to a position that is defined relative to this current position. The figure shows an example of relative (command **MOVE**) and absolute (command **MOVEABS**) linear moves.

1-3-1 PTP control

In point-to-point positioning, each axis is moved independently of the other axis. The CJ1W-MCH72 supports the following operations:

- Relative move
- Absolute move
- Continuous move forward
- Continuous move reverse.

1-3-1-1 Relative and absolute moves



To move a single axis either the command **MOVE** for a relative move or the command **MOVEABS** for an absolute move is used. Each axis has its own move characteristics, which are defined by the axis parameters.

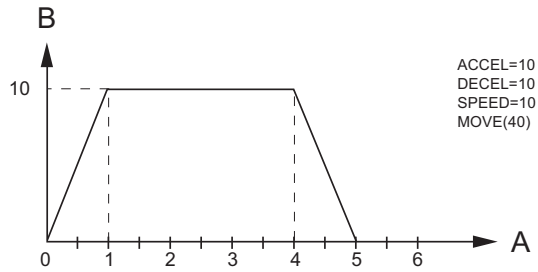
Suppose a control program is executed to move from the origin to an axis no. 0 (A) coordinate of 100 and axis no. 1 (B) coordinate of 50. If the speed parameter is set to be the same for both axes and the acceleration and deceleration rate are set sufficiently high, the movements for axis 0 and axis 1 will be as shown in the figure.

At start, both the axis 0 and axis 1 moves to a coordinate of 50 over the same duration of time. At this point, axis 1 stops and axis 0 continues to move to a coordinate of 100.

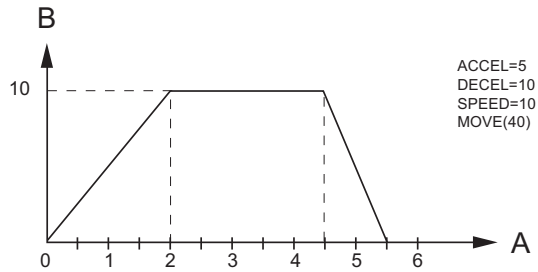
The move of a certain axis is determined by the axis parameters. Some relevant parameters are:

Parameter	Description
UNITS	Unit conversion factor
ACCEL	Acceleration rate of an axis in units/s^2
DECEL	Deceleration rate of an axis in units/s^2
SPEED	Demand speed of an axis in units/s^2

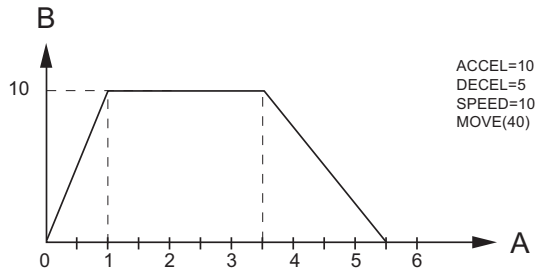
Defining moves



The speed profile in this figure shows a simple **MOVE** operation. Axis A is the time, axis B is the speed. The **UNITS** parameter for this axis has been defined for example as meters. The required maximum speed has been set to 10 m/s. In order to reach this speed in one second and also to decelerate to zero speed again in one second, both the acceleration as the deceleration rate have been set to 10 m/s^2. The total distance travelled is the sum of distances travelled during the acceleration, constant speed and deceleration segments. Suppose the distance moved by the **MOVE** command is 40 m, the speed profile is given by the figure.



The two speed profiles in these figures show the same movement with an acceleration time respectively a deceleration time of 2 seconds. Again, Axis A is the time, axis B is the speed.



Move calculations

The following equations are used to calculate the total time for the motion of the axes.

- The moved distance for the **MOVE** command is *D*.
- The demand speed is *V*.
- The acceleration rate is *a*.
- The deceleration rate is *d*.

Acceleration time = $\frac{V}{a}$

Acceleration distance = $\frac{V^2}{2a}$

Deceleration time = $\frac{V}{d}$

Deceleration distance = $\frac{V^2}{2d}$

Constant speed distance = $D - \frac{V^2(a+d)}{2ad}$

Total time = $\frac{D}{V} + \frac{V(a+d)}{2ad}$

1-3-1-2 Continuous moves

The **FORWARD** and **REVERSE** commands can be used to start a continuous movement with constant speed on a certain axis. The **FORWARD** command moves the axis in positive direction and the **REVERSE** command in negative direction. For these commands also the axis parameters **ACCEL** and **SPEED** apply to specify the acceleration rate and demand speed.

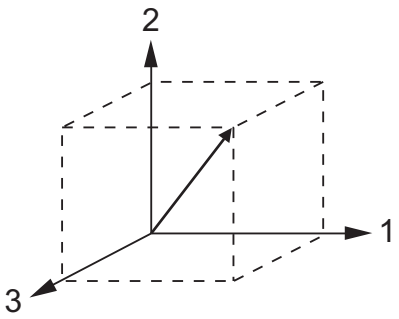
Both movements can be cancelled by using either the **CANCEL** or **RAPIDSTOP** command. The **CANCEL** command cancels the move for one axis and **RAPIDSTOP** cancels moves on all axes. The deceleration rate is set by **DECEL**.

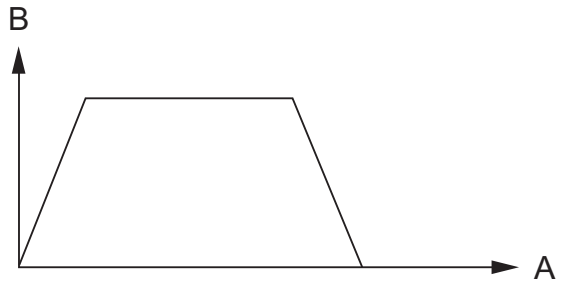
1-3-2 CP control

Continuous Path control enables to control a specified path between the start and end position of a movement for one or multiple axes. The CJ1W-MCH72 supports the following operations:

- Linear interpolation
- Circular interpolation
- CAM control.

1-3-2-1 Linear interpolation



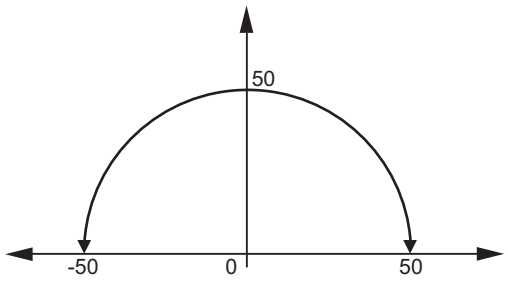


In applications it can be required for a set of motors to perform a move operation from one position to another in a straight line. Linearly interpolated moves can take place among several axes. The commands **MOVE** and **MOVEABS** are also used for the linear interpolation. In this case the commands will have multiple arguments to specify the relative or absolute move for each axis.

Consider the three axis move in a 3-dimensional plane in the figure. It corresponds to the **MOVE(50,50,50)** command. The speed profile of the motion along the path is given in the diagram. The three parameters **SPEED**, **ACCEL** and **DECEL** that determine the multi axis movement are taken from the corresponding parameters of the base axis. The **MOVE** command computes the various components of speed demand per axis.

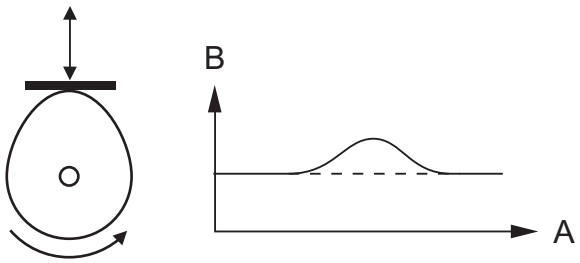
A is the time axis, B is the speed axis.

1-3-2-2 Circular interpolation



It may be required that a tool travels from the starting point to the end point in an arc of a circle. In this instance the motion of two axes is related via a circular interpolated move using the **MOVECIRC** command.
Consider the diagram in the figure. It corresponds to the **MOVECIRC(-100,0,-50,0,0)** command. The centre point and desired end point of the trajectory relative to the start point and the direction of movement are specified. The **MOVECIRC** command computes the radius and the angle of rotation. Like the linearly interpolated **MOVE** command, the **ACCEL**, **DECEL** and **SPEED** variables associated with the base axis determine the speed profile along the circular move.

1-3-2-3 CAM control



Additional to the standard move profiles the CJ1W-MCH72 also provides a way to define a position profile for the axis to move. The **CAM** command moves an axis according to position values stored in the CJ1W-MCH72 Table array. The speed of travelling through the profile is determined by the axis parameters of the axis.
The figure corresponds to the command **CAM(0,99,100,20)**. A is the time axis, B is the position axis.

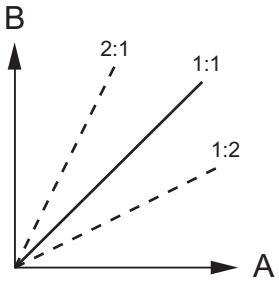
1-3-3 EG control

Electronic Gearing control allows you to create a direct gearbox link or a linked move between two axes. The MC Unit supports the following operations.

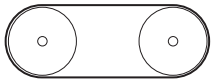
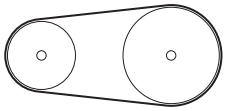
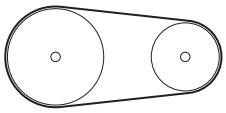
- Electronic gearbox
- Linked CAM

- Linked move
- Adding axes

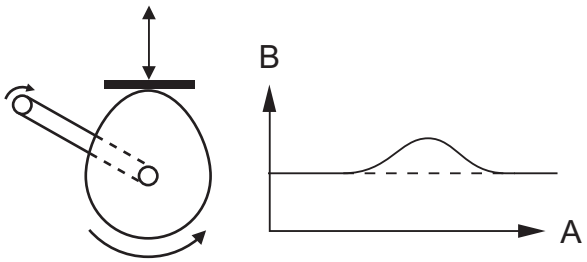
1-3-3-1 Electronic gearbox



The CJ1W-MCH72 is able to have a gearbox link from one axis to another as if there is a physical gearbox connecting them. This can be done using the **CONNECT** command in the program. In the command the ratio and the axis to link to are specified.
In the figure, A is the Master axis, and B is the **CONNECT** axis.

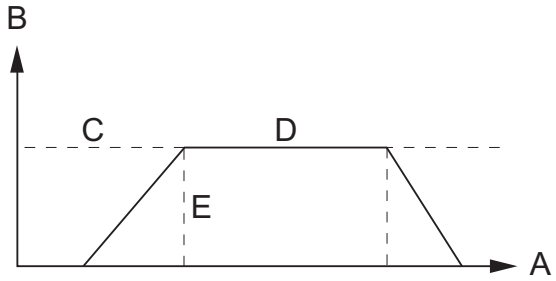
Axes		Ratio	CONNECT command
0	1		
		1:1	**CONNECT(1,0) AXIS(1)**
		2:1	**CONNECT(2,0) AXIS(1)**
		1:2	**CONNECT(0.5,0) AXIS(1)**

1-3-3-2 Linked CAM control



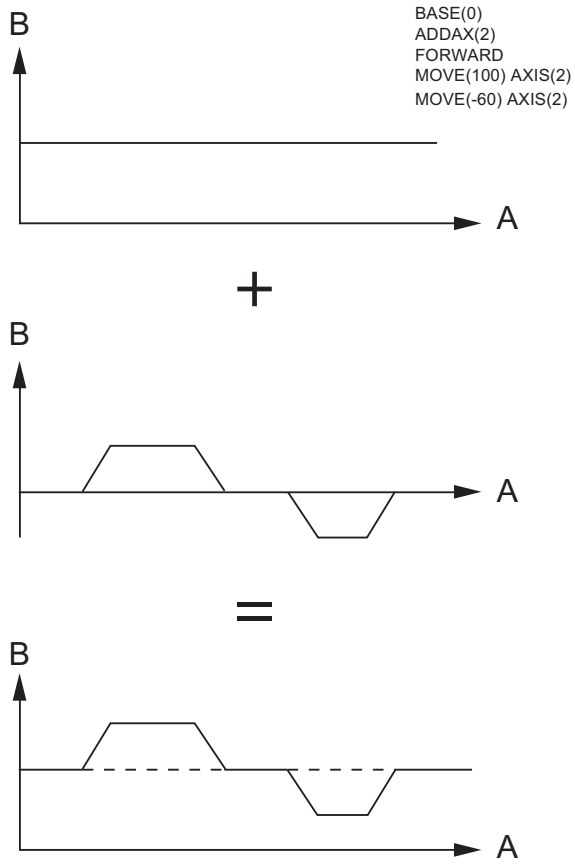
Next to the standard CAM profiling tool the CJ1W-MCH72 also provides a tool to link the CAM profile to another axis. The command to create the link is called **CAMBOX**. The travelling speed through the profile is not determined by the axis parameters of the axis but by the position of the linked axis. This is like connecting two axes through a cam.
In the figure, A is the Master axis (0) position, and B is the **CAMBOX** Axis (1) position.

1-3-3-3 Linked move



The **MOVELINK** command provides a way to link a specified move to a master axis. The move is divided into an acceleration, deceleration and constant speed part and they are specified in master link distances. This can be particularly useful for synchronizing two axes for a fixed period.
The labels in the figure are:
A Time axis.
B Speed axis.
C Master axis (1).
D Synchronized.
E **MOVELINK** axis (0).

1-3-3-4 Adding axes



It is very useful to be able to add all movements of one axis to another. One possible application is for instance changing the offset between two axes linked by an electronic gearbox. The CJ1W-MCH72 provides this possibility by using the **ADDAX** command. The movements of the linked axis will consists of all movements of the actual axis plus the additional movements of the master axis.
In the figure, A is the time axis and B is the speed axis.

1-3-4 Other operations

1-3-4-1 Cancelling moves

In normal operation or in case of emergency it can be necessary to cancel the current movement from the buffers. When the **CANCEL** or **RAPIDSTOP** commands are given, the selected axis respectively all axes will cancel their current move.

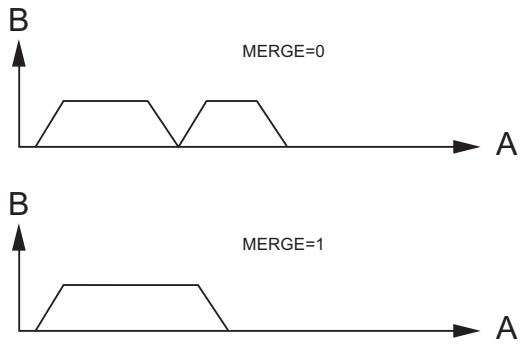
1-3-4-2 Origin search

The encoder feedback for controlling the position of the motor is incremental. This means that all movement must be defined with respect to an origin point. The **DATUM** command is used to set up a procedure whereby the CJ1W-MCH72 goes through a sequence and searches for the origin based on digital inputs and/or Z-marker from the encoder signal.

1-3-4-3 Print registration

The CJ1W-MCH72 can capture the position of an axis in a register when an event occurs. The event is referred to as the print registration input. On the rising or falling edge of an input signal, which is either the Z-marker or an input, the CJ1W-MCH72 captures the position of an axis in hardware. This position can then be used to correct possible error between the actual position and the desired position. The print registration is set up by using the **REGIST** command.
The position is captured in hardware, and therefore there is no software overhead and no interrupt service routines, eliminating the need to deal with the associated timing issues.

1-3-4-4 Merging moves



If the **MERGE** axis parameter is set to 1, a movement is always followed by a subsequent movement without stopping. The figures show the transitions of two moves with **MERGE** value 0 and value 1.
In the figure, A is the time axis and B is the speed axis.

1-3-4-5 Jogging

Jogging moves the axes at a constant speed forward or reverse by manual operation of the digital inputs. Different speeds are also selectable by input. Refer to the **FWD_JOG**, **REV_JOG** and **FAST_JOG** axis parameters.

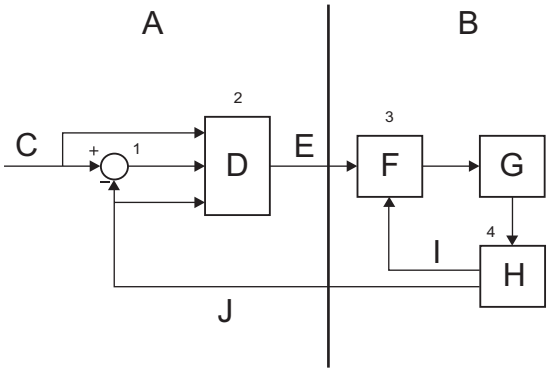
1-4 Servo system principles

The servo system used by and the internal operation of the CJ1W-MCH72 are briefly described in this section.

1-4-1 Semi-closed loop system

The servo system of the CJ1W-MCH72 uses a semi-closed or inferred closed loop system. This system detects actual machine movements by the rotation of the motor in relation to a target value. It calculates the error between the target value and actual movement, and reduces the error through feedback.

1-4-2 Internal operation of the CJ1W-MCH72



Inferred closed loop systems occupy the mainstream in modern servo systems applied to positioning devices for industrial applications. The figure shows the basic principle of the servo system as used in the CJ1W-MCH72.

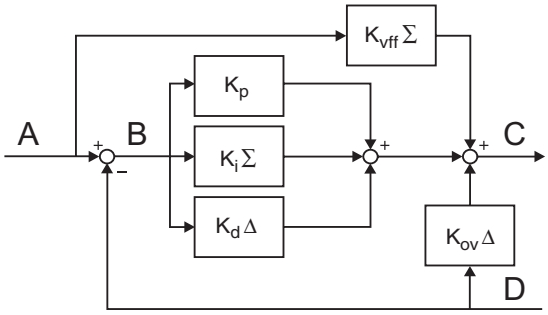
1 The CJ1W-MCH72 performs actual position control. The main input of the controller is the Following Error, which is the calculated difference between the demand position and the actual measured position.
2 The Position Controller calculates the required speed reference output determined by the Following Error and possibly the demanded position and the measured position. The speed reference is provided to the Servo Driver.
3 The Servo Driver controls the rotational speed of the servo motor corresponding to the speed reference. The rotational speed is proportional to the speed reference.
4 The rotary encoder generates the feedback pulses for both the speed feedback within the Servo Driver speed loop and the position feedback within the CJ1W-MCH72 position loop.

The labels in the figure are:

A CJ1W-MCH72.
B Servo system.
C Demand position.
D Position control.
E Speed reference.
F Speed control.
G Motor.
H Encoder.
I Measured speed.
J Measured position.

1-4-3 Motion control algorithm

The servo system controls the motor by continuously adjusting the speed reference to the Servo Driver. The speed reference is calculated by the motion control algorithm of the CJ1W-MCH72, which is explained in this section.



The motion control algorithm uses the demand position (A), the measured position (D) and the Following Error (B) to determine the speed reference. The Following Error is the difference between the demanded and measured position. The demand position, the measured position and the Following Error are represented by the axis parameters **MPOS**, **DPOS** and **FE**. Five gain values have been implemented for the user to be able to configure the correct control operation for each application.
C is the output signal.

- Proportional gain
  The proportional gain K_p creates an output O_p that is proportional to the Following Error E.
  $O_p = K_p \cdot E$
  All practical systems use proportional gain. For many just using this gain parameter alone is sufficient. The proportional gain axis parameter is called **P_GAIN**.
- Integral gain
  The integral gain K_i creates an output O_i that is proportional to the sum of the Following Errors that have occurred during the system operation.
  $O_i = K_i \cdot \Sigma E$
  Integral gain can cause overshoot and so is usually used only on systems working at constant speed or with slow accelerations. The integral gain axis parameter is called **I_GAIN**.
- Derivative gain
  The derivative gain K_d produces an output O_d that is proportional to the change in the Following Error E and speeds up the response to changes in error while maintaining the same relative stability.
  $O_d = K_d \cdot \Delta E$
  Derivative gain may create a smoother response. High values may lead to oscillation. The derivative gain axis parameter is called **D_GAIN**.
- Output speed gain
  The output speed gain K_{ov} produces an output O_{ov} that is proportional to the change in the measured position P_m and increases system damping.
  $O_{ov} = K_{ov} \cdot \Delta P_m$

The output speed gain can be useful for smoothing motions but will generate high Following Errors. The output speed gain axis parameter is called **OV_GAIN**.

- Speed feed forward gain
  The speed feedforward gain K_{vff} produces an output O_{vff} that is proportional to the change in demand position P_d and minimizes the Following Error at high speed.
  $O_{vff} = K_{vff} \cdot \Delta P_d$
  The parameter can be set to minimise the Following Error at a constant machine speed after other gains have been set. The speed feed forward gain axis parameter is called **VFF_GAIN**.

The default settings are given in the table along with the resulting profiles. Fractional values are allowed for gain settings.

Gain	Default value
Proportional gain	0.1
Integral gain	0.0
Derivative gain	0.0
Output speed gain	0.0
Speed feedforward gain	0.0

1-5 Trajexia system architecture

The system architecture of the Trajexia is dependant upon these concepts:

- Program control
- Motion Sequence
- Motion buffers
- Communication
- Peripherals

These concepts depend upon the value set in the **SERVO_PERIOD** parameter. The relationship between the value of **SERVO_PERIOD** and the different concepts of the system architecture are describes as follows.

1-5-1 Program control

Programs make the system work in a defined way. The programs are written in a language similar to BASIC and control the application of the axes and modules. 14 Programs can be executed in parallel. The programs can be set to run at system power-up, started and stopped from other programs and executed from Trajexia Studio.
Programs execute commands to move the axes, control inputs and outputs and make communication via BASIC commands.

1-5-2 Motion sequence

The motion sequence controls the position of all 32 axes with the actions as follows:

- Reading the Motion buffer
- Reading the current Measured Position (MPOS)
- Calculating the next Demanded Position (DPOS)
- Executing the Position loop
- Sending the Axis reference
- Error handling

1-5-3 Motion buffers

Motion buffers are the link between the BASIC commands and the Axis control loop. When a BASIC motion command is executed, the command is stored in one of the buffers. During the next motion sequence, the profile generator executes the movement according to the information in the buffer.
When the movement is finished, the motion command is removed from the buffer.

1-5-4 Communication

The CJ1W-MCH72 can exchange data with memory areas in the PLC. This enables the CJ1W-MCH72 to use the inputs and outputs connected to the PLC. Also, programs in the CJ1W-MCH72 and PLC programs can exchange control and status data.
For more information on communication and data exchange, refer to chapter 3.

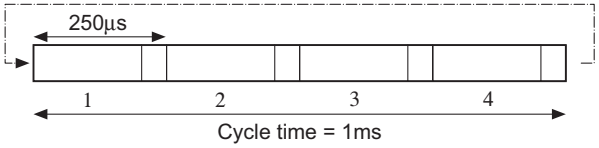
1-5-5 Peripherals

All inputs and outputs are used with the set of parameters (**IN**, **OP**, **AIN**, **AOUT**). The inputs and outputs are automatically detected and mapped in Trajexia. Inverters are considered a peripheral device and have a set of BASIC commands to control them.

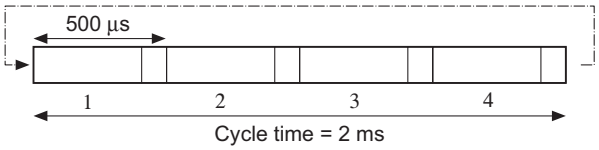
1-6 Cycle time

All processes in the Trajexia system are based on the cycle time. The cycle time is divided into four CPU tasks:

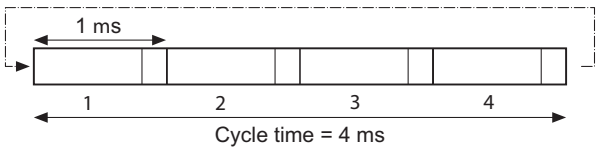
- 250 μs time intervals for a **SERVO_PERIOD** of 0.5 and 1.0 ms



- 500 μs time intervals for a **SERVO_PERIOD** of 2.0 ms



- 1 ms time intervals for a **SERVO_PERIOD** of 4.0 ms



The processes that can be carried out in each time interval depends on the **SERVO_PERIOD** that is set.

The operations executed in each CPU task are:

CPU task	Operation
First CPU task	Motion Sequence Low priority process
Second CPU task	High priority process
Third CPU task	Motion Sequence (only if **SERVO_PERIOD**=0.5ms) LED Update. High priority process
Fourth CPU task	External Communications

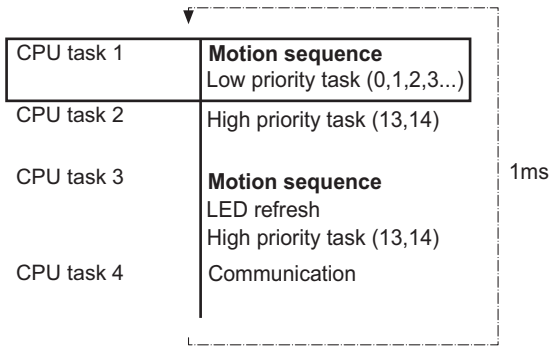
Note The Motion sequence execution depends on setting of the **SERVO_PERIOD** parameter.

1-6-1 Servo period

The **SERVO_PERIOD** can be set at 0.5, 1, 2 or 4 ms. The processes that take place within the cycle time depend on the setting of the **SERVO_PERIOD** parameter. The **SERVO_PERIOD** parameter is a Trajexia parameter that must be set according to the system configuration.
The factory setting is 1ms (**SERVO_PERIOD=1000**). A change is set only after a restart of the CJ1W-MCH72.

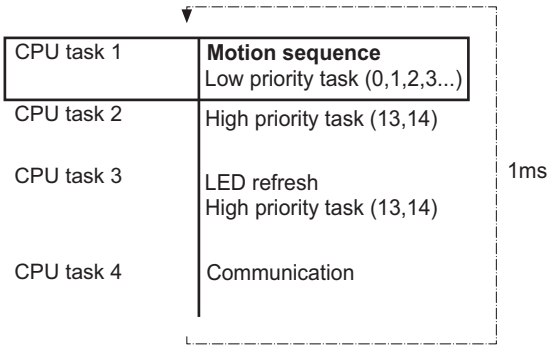
Note Only the Sigma-III Servo Driver and the Sigma-V Servo Driver support the 0.5 ms transmission cycle.

1-6-1-1 Servo period 0.5 ms



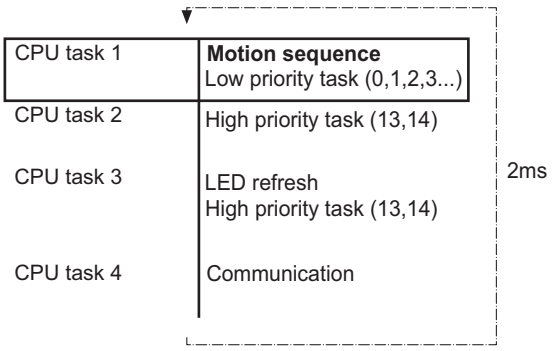
The **SERVO_PERIOD** has a value of 0.5ms and the motion sequence is executed every 0.5ms.

1-6-1-2 Servo period 1 ms



The **SERVO_PERIOD** has a value of 1ms and the motion sequence is executed every 1ms. As the motion sequence is not executed during CPU task 3, there is more time for the program execution. High priority programs run faster.

1-6-1-3 Servo period 2 ms



The **SERVO_PERIOD** has a value of 2ms and the motion sequence is executed every 2.0ms.

1-6-1-4 Servo period rules

The number of axes and MECHATROLINK-II slaves in the Trajexia system determines the value of the **SERVO_PERIOD** system parameter.
There are 2 types of MECHATROLINK-II slaves that are supported by the CJ1W-MCH72 units:

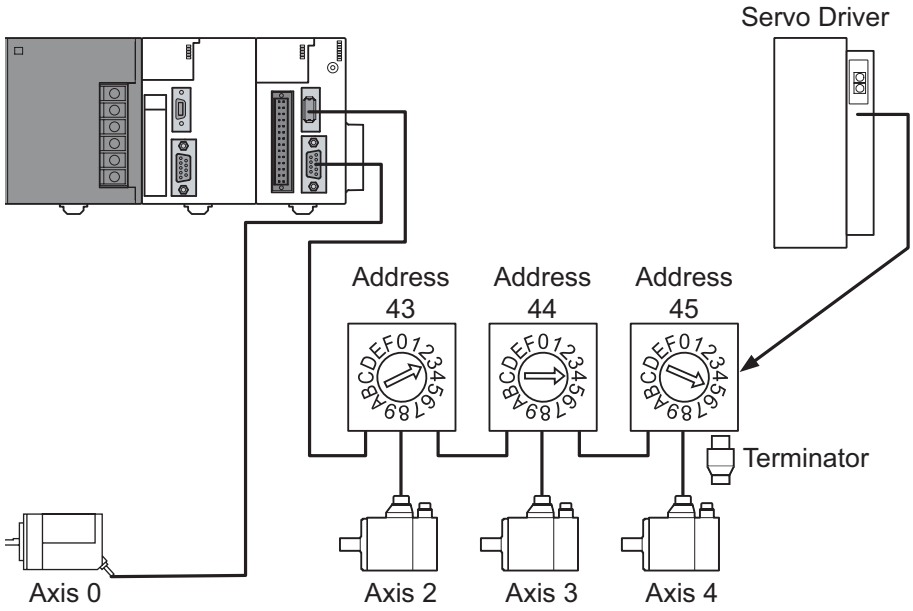
- Servo Drivers
  The CJ1W-MCH72 considers Servo Drivers as axes.
- Inverters
  The CJ1W-MCH72 does not consider Inverters as axes.

You must obey the most restrictive rules when you set the **SERVO_PERIOD** parameter. An incorrect value of the **SERVO_PERIOD** parameter results in an incorrect detection of the MECHATROLINK-II slaves.
The most restrictive rules are given in the tables below. For each unit the table lists the maximum number of slaves the unit can control at the given **SERVO_PERIOD** setting.

SERVO_PERIOD	Highest axis number	Max. number of MECHATROLINK-II axes	Max. number of MECHATROLINK-II inverters
0.5 ms	7	4	4
1.0 ms	15	8	8
2.0 ms	31	16	8
4.0 ms	31	30	8

1-6-1-5 Configuration examples

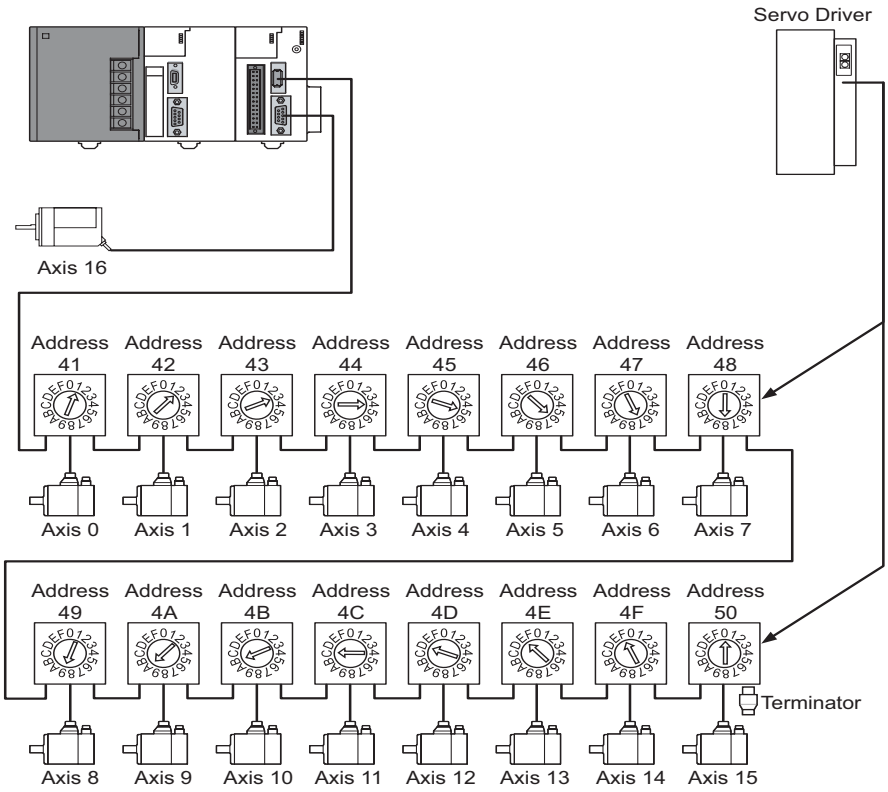
Example 1



- 1x CJ1W-MCH72
- 3x Sigma-II Servo Driver
- 1x Encoder (Axis 0)
- **SERVO_PERIOD** = 1ms

The CJ1W-MCH72 supports 1.0ms **SERVO_PERIOD** with 4 axes.

Example 2



- 1x CJ1W-MCH72
- 16x Sigma-II Servo Driver
- 1x Encoder (Axis 16)
- **SERVO_PERIOD** = 2ms

The CJ1W-MCH72 supports 2ms **SERVO_PERIOD** with 17 axes.

1-7 Program control and multi-tasking

The Trajexia system has program, processes and multi tasking control.

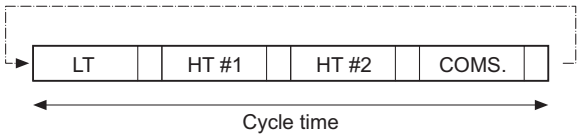
1-7-1 Program control

The Trajexia system can control 14 processes that are written as BASIC programs. When the program is set to run, the program is executed.
Processes 1 to 12 are low priority, 13 and 14 are high priority.

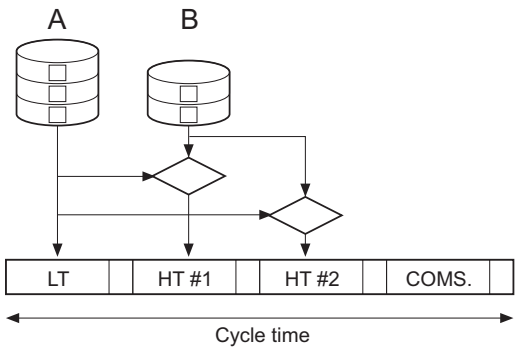
1-7-2 Processes

The low-priority process 0 is reserved for the Terminal window of Trajexia Studio. This terminal window is used to write direct BASIC commands to the CJ1W-MCH72 independent to other programs. These commands are executed after you press the Enter button.

1-7-3 Multi-tasking

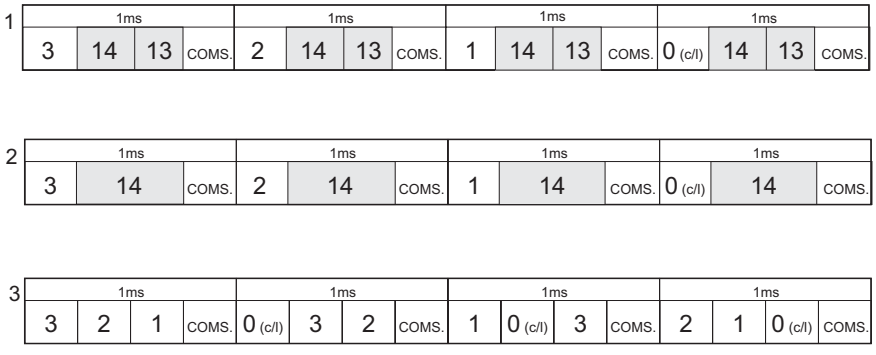


Each cycle time is divided into 4 time slices called CPU tasks. Processes run in the first 3 CPU tasks according to the priority of the process.
Motion sequence and low-priority processes (A) are executed in the Low Task (LT) period.
High priority processes (B) are executed in the high Task (HT) periods.



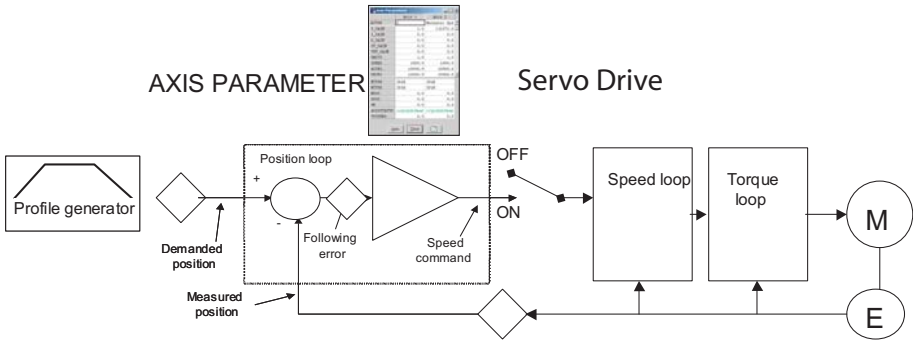
External communication that are not related to the motion network are updated in the communications (COMS) period in the fourth CPU task.
Trajexia can control up to 14 programs at the same time.
In contrast to low priority processes, a high priority process is always available for execution during two of the four CPU tasks. The high-priority tasks are executed faster than the low-priority tasks, it is that they have more time available for their execution. All the low-priority tasks must share one slot of time and the high-priority task have their own two slots of time.

1-7-4 Multi-tasking example



In the example 1, there are two high-priority processes, 13 and 14. The two HT periods are reserved for these processes, one for processes 13 and one for processes 14. The low-priority processes 3, 2, 1 and 0 are executed in the LT period, one process per Cycle time here set to 1.0ms.
In the middle example, there is only one high-priority process, 14. Both HT periods are reserved for this process. The low-priority processes, 3, 2, 1 and 0 are executed in the LT period, one process per cycle time.
In the lower example, there are no high-priority processes. Therefore, the HT periods can be used for the low-priority processes. The LT period is also used for the low-priority processes.

1-8 Motion sequence and axes



Motion sequence is the part of the CJ1W-MCH72 that controls the axes. The actual way that the motion sequence operates depends on the axis type. The axis type can be set and read by the parameter **ATYPE**. At start-up the Trajexia system automatically detects the configuration of the axes.

- The default value for the parameter **ATYPE** for MECHATROLINK-II axes is 41 (MECHATROLINK-II speed).
- The default value for the parameter **ATYPE** for the Encoder Interface is 44 (incremental encoder).

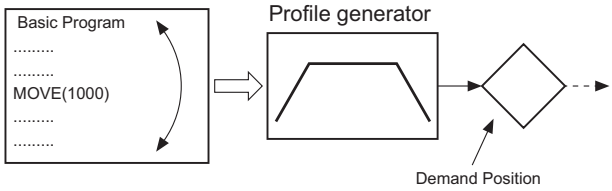
All non allocated axes are set as a virtual axis. The value for the parameter **ATYPE** is 0.
Every axis has the general structure as shown in .

The motion sequence which will be executed at the beginning of each servo period will contain the following elements:

1 Transfer any moves from BASIC process buffers to motion buffers (see section 1-9).
2 Read digital inputs.
3 Load moves. (See note.)
4 Calculate speed profile. (See note.)
5 Calculate axis positions. (See note.)
6 Execute position servo. For axis 0 this also includes the Servo Driver communications. (See note.)
7 Update outputs.

Note Each of these items will be performed for each axis in turn before moving on to the next item.

1-8-1 Profile generator



The profile generator is the algorithm that calculates the demanded position for each axis. The calculation is made every motion sequence.
The profile is generated according to the motion instructions from the BASIC programs.

1-8-2 Position loop

The position loop is the algorithm that makes sure that there is a minimal deviation between the measured position (**MPOS**) and the demand position (**DPOS**) of the same axis.

1-8-3 Axis sequence

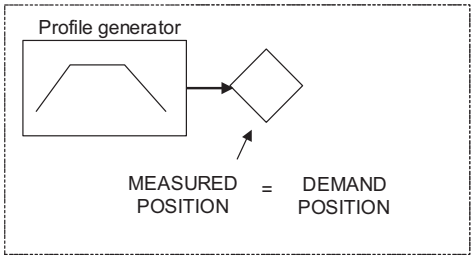
- The motion controller applies motion commands to an axis array that is defined with the **BASE** command. If the motion command concerns one axis, it is applied to the first axis in the **BASE** array. If the motion command concerns more than one axis, and makes an orthogonal move, the axes are taken from the array in the order defined by the **BASE** command. For more information on the **BASE** command and the definition of the axis sequence in an axis array, refer to the Trajexia Programming Manual, chapter 3 (BASIC commands).
- If **SERVO=OFF** for one axis, the motion commands for that axis are ignored.
- If the Following Error (**FE**) in one axis exceeds the parameter value **FELIMIT**, the next action occurs:
  - **WDOG** is set to **OFF** and all axes stop.
  - **SERVO** for the axis that causes the error goes to **OFF**.
  - The current move is cancelled and removed from the buffer.

1-8-4 Type of axis

ATYPE	Applicable to	Name	Description
0	All axes	Virtual axis	Internal axis with no physical output. It is the only valid setting for non-allocated axes. That is, those that are not MECHATROLINK-II Servo Drivers.

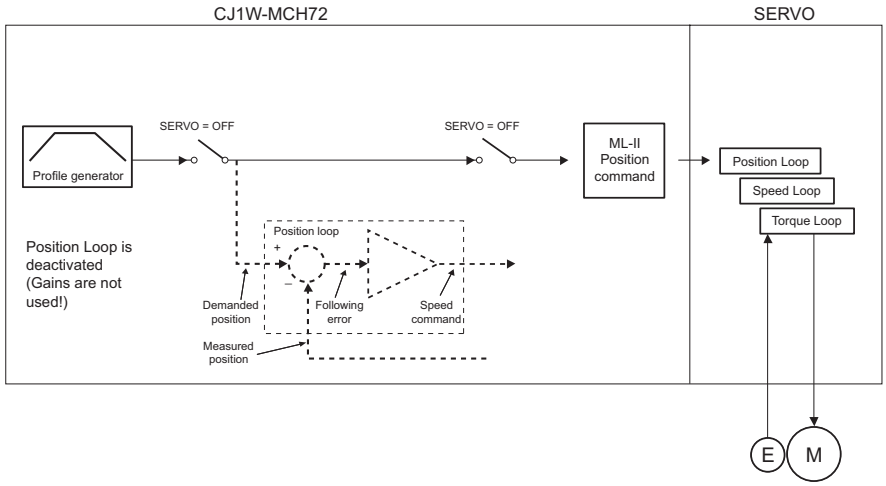
ATYPE	Applicable to	Name	Description
40	MECHATROLINK-II Servo Drivers	MECHATROLINK-II Position	Position loop in the Servo Driver. CJ1W-MCH72 sends position reference to the Servo Driver via MECHATROLINK-II.
41		MECHATROLINK-II Speed (Default)	Position loop in the Trajexia. CJ1W-MCH72 sends speed reference to the Servo Driver via MECHATROLINK-II.
42		MECHATROLINK-II Torque	Position loop in the Trajexia. CJ1W-MCH72 sends torque reference to the Servo Driver via MECHATROLINK-II.
43	External driver connected to encoder input	Stepper output	Pulse and direction outputs. Position loop is in the driver. CJ1W-MCH72 sends pulses and receives no feed back.
44		Servo axis (Default) Encoder	CJ1W-MCH72 receives position from an incremental encoder.
45		Encoder output	The same as stepper, but with the phase differential outputs emulating an incremental encoder.
47		Absolute EnDat	Feedback is received from an EnDat absolute encoder.
48		Absolute SSI	Feedback is received from an SSI absolute encoder.
49	MECHATROLINK-II Inverters	Inverter as axis	Inverters (with built-in encoder interface) are controlled on the MECHATROLINK-II bus as servo axes.

1-8-4-1 Virtual axis ATYPE=0



You can split a complex profile into two or more simple movements, each assigned to a virtual axis. These movements can be added together with the BASIC command **ADDAX** then assigned to a real axis.

1-8-4-2 MECHATROLINK-II position ATYPE=40

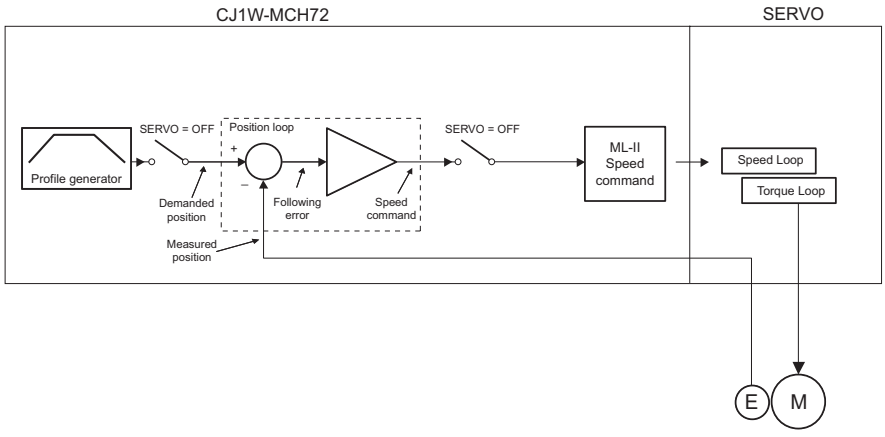


With **SERVO = ON**, the position loop is closed in the Servo Driver. Gain settings in the CJ1W-MCH72 have no effect. The position reference is sent to the Servo Driver.

Note Although **MPOS** and **FE** are updated, the real value is the value in the Servo Driver. The real Following Error can be monitored by the **DRIVE_MONITOR** parameter by setting **DRIVE_CONTROL = 2**.

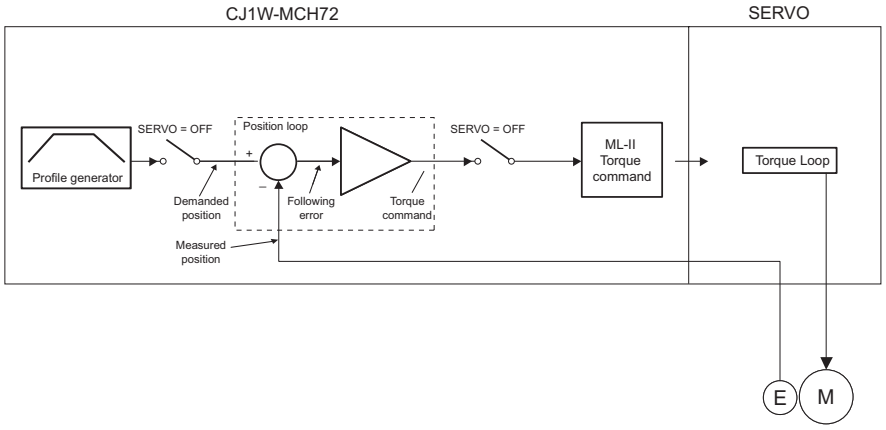
Note The MECHATROLINK-II position **ATYPE = 40** is the recommended setting to obtain a higher performance of the servo motor.

1-8-4-3 MECHATROLINK-II speed ATYPE=41



With **SERVO = ON**, the speed loop is closed in the CJ1W-MCH72.
Speed reference is sent to the Servo Driver. This setting is not recommended, since there is one cycle delay in the loop (DPOS(n) is compared with MPOS(n-1)).
With **SERVO = OFF**, the speed reference is sent via **S_REF** command. 0x40000000 means maximum speed of the servomotor. This is the recommended setting.

1-8-4-4 MECHATROLINK-II torque ATYPE=42



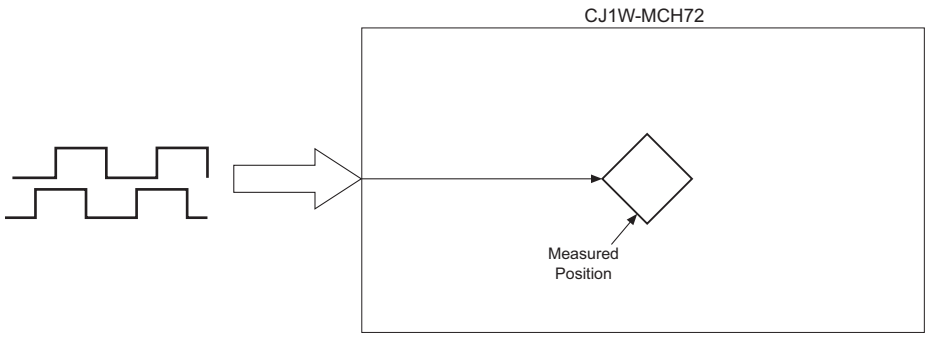
With **SERVO = ON**, the torque loop is closed in the CJ1W-MCH72. The torque reference in the Servo Driver depends on the **FE** and the gain.
With **SERVO = OFF**, the torque reference is sent directly via the **T_REF** command. 0x40000000 is the maximum torque of the servomotor.

Note To monitor the torque in the servo in **DRIVE_MONITOR**, set **DRIVE_CONTROL=11**.

1-8-4-5 Stepper output ATYPE=43

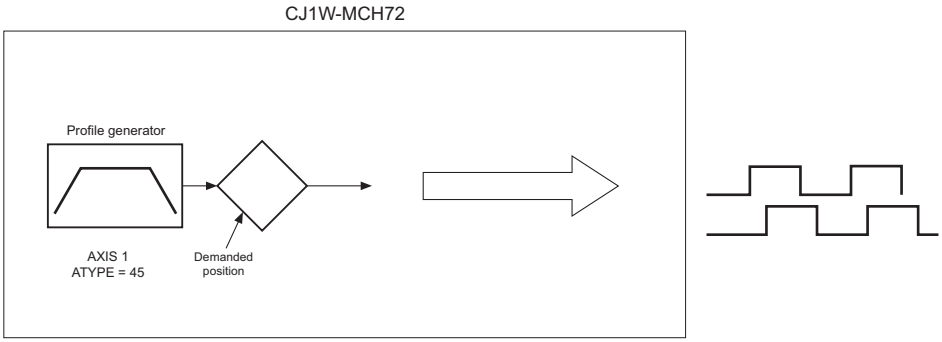
The position profile is generated and the output from the system is a pulse train and direction signal. This is useful to control a motor via pulses or as a position reference for another motion controller.

1-8-4-6 Servo axis ATYPE=44



With **SERVO = OFF**, the position of the external incremental encoder is read.

1-8-4-7 Encoder output ATYPE=45



The position profile is generated and the output from the system is an incremental encoder pulse. This is useful to control a motor via pulses or as a position reference for another motion controller.

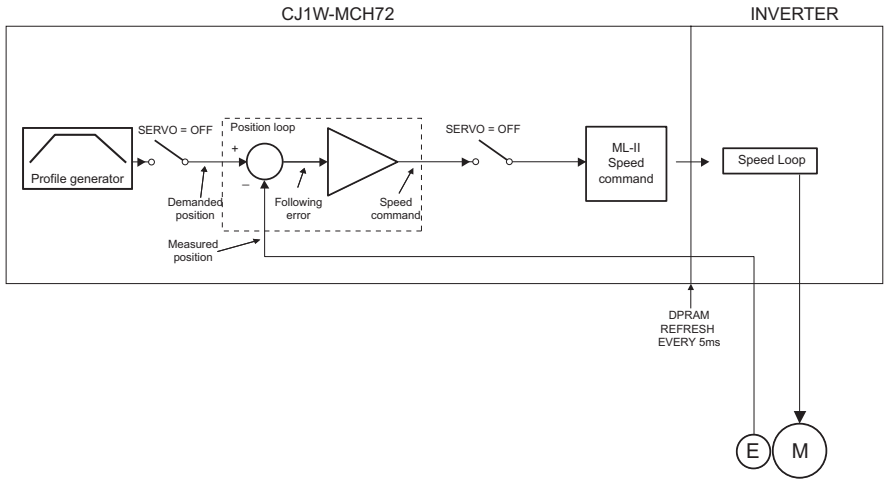
1-8-4-8 Absolute EnDat encoder ATYPE=47

With **SERVO = OFF**, the position of the external absolute EnDat encoder is read.

1-8-4-9 Absolute SSI encoder ATYPE=48

With **SERVO = OFF**, the position of the external absolute SSI encoder is read.

1-8-4-10 Inverter axis ATYPE=49



This type allows Inverters (with built-in encoder interface) to be controlled on the MECHATROLINK-II bus as servo axes.
From the controller point of view, Inverter axes are handled the same as servo axes in MECHATROLINK-II Speed Mode (**ATYPE=44**).
Unlike the other axis types, this Inverter axis must be defined programmatically with function 8 of the command **INVERTER_COMMAND**.

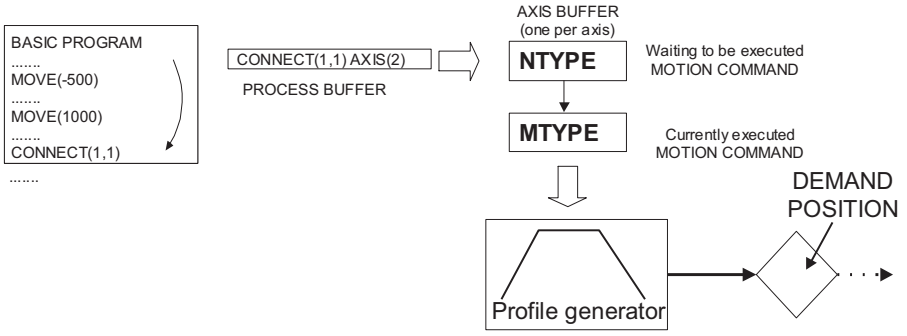
The Speed command to the Inverter and the feedback from the encoder is refreshed in the Inverter every 5 ms. This is a DPRAM limitation. This means that the use of the Inverter is similar to the use of a Servo Driver, but the performance is lower.

1-8-4-11 Summary of axis types and control modes

The following table lists the axis types and their recommended modes for speed control, position control and torque control.

ATYPE	SERVO	Mode	Comment
40	OFF	Position (MECHATROLINK-II)	The position loop is closed in the Servo Driver. No new motion command is allowed.
40	ON	Position (MECHATROLINK-II)	Recommended mode for position control with MECHATROLINK-II axes.
41	OFF	Speed (MECHATROLINK-II)	Recommended mode for speed control with MECHATROLINK-II axes. Set the speed with **S_REF**.
41	ON	Position (MECHATROLINK-II)	The position loop is closed in Trajexia. This gives lower performance than closing the position loop in the Servo Driver.
42	OFF	Torque (MECHATROLINK-II)	Recommended mode for torque control with MECHATROLINK-II axes. Set the torque with **T_REF**.
42	ON	Position via torque (MECHATROLINK-II)	The position loop is closed in Trajexia. The output of the position loop is sent as the torque reference to the Servo Driver.
49	OFF	Speed	Inverter (with built-in encoder interface) controlled on the MECHATROLINK-II bus as a servo axis. Set the speed with **S_REF**.
49	ON	Position	Inverter (with built-in encoder interface) controlled on the MECHATROLINK-II bus as a servo axis. The position loop is closed in Trajexia.

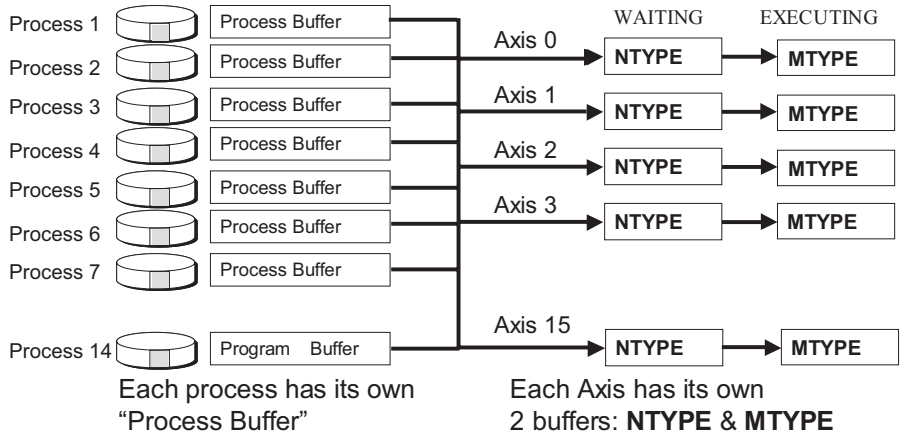
1-9 Motion buffers



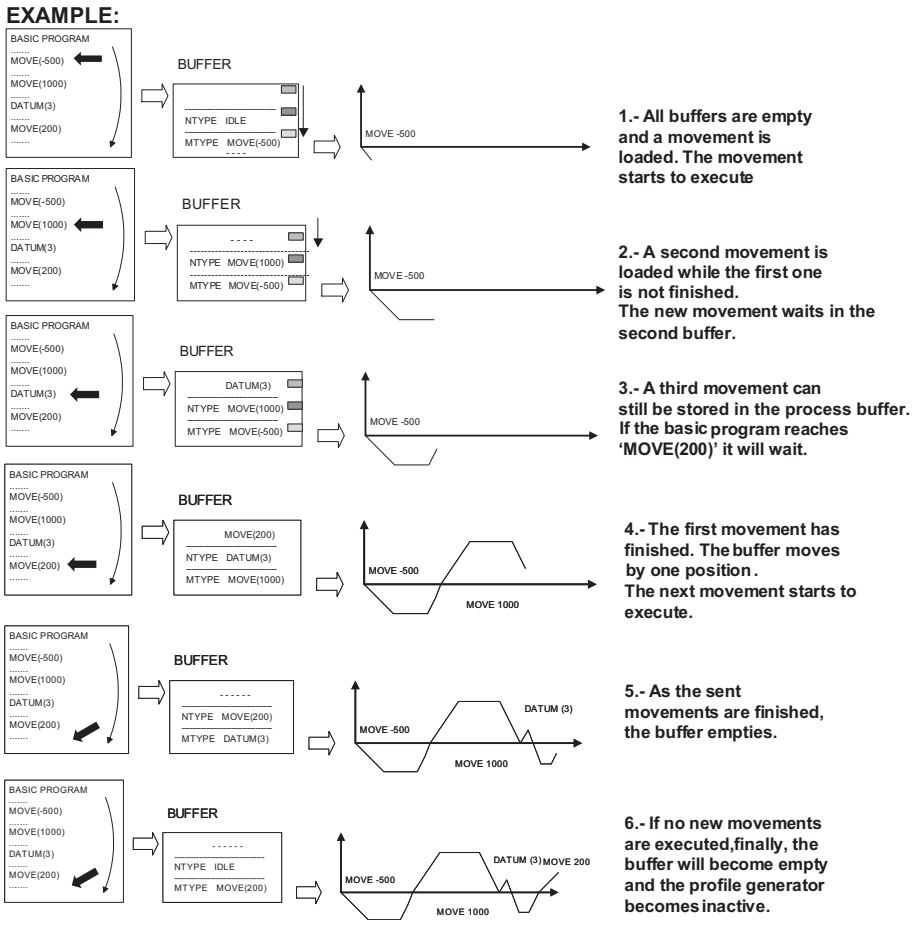
The motion buffer is a temporary store of the motion instruction from the BASIC program to the profile generator.
The BASIC program continues while the instruction waits in the buffer.
There are three types of buffer:

- **MTYPE**. The current movement that is being executed. **MTYPE** relates to the axis and not to the process.
- **NTYPE**. The new movement that waits for execution. **NTYPE** relates to the axis and not to the process.
- Process Buffer. The third buffered movement cannot be monitored. The process buffer relates to the process and not to the axis.

It is possible to check if the process buffer is full by checking the **PMOVE** process parameter.



When a motion instruction is executed in the BASIC program, the instruction is loaded into the process buffer and distributed to the corresponding axis buffer in the next motion sequence.
If a fourth motion instruction is executed and the three buffers are full, the BASIC program stops execution until a process buffer is free for use.


EXAMPLE:
BASIC PROGRAM
MOVE(-500)
MOVE(1000)
DATUM(3)
MOVE(200)
BUFFER
NTYPE IDLE
MTYPE MOVE(-500)
MOVE -500
1.- All buffers are empty and a movement is loaded. The movement starts to execute
NTYPE MOVE(1000)
MTYPE MOVE(-500)
2.- A second movement is loaded while the first one is not finished. The new movement waits in the second buffer.
DATUM(3)
NTYPE MOVE(1000)
MTYPE MOVE(-500)
3.- A third movement can still be stored in the process buffer. If the basic program reaches 'MOVE(200)' it will wait.
MOVE(200)
NTYPE DATUM(3)
MTYPE MOVE(1000)
MOVE -500
MOVE 1000
4.- The first movement has finished. The buffer moves by one position. The next movement starts to execute.
NTYPE MOVE(200)
MTYPE DATUM(3)
DATUM (3)
5.- As the sent movements are finished, the buffer empties.
NTYPE IDLE
MTYPE MOVE(200)
MOVE 200
6.- If no new movements are executed,finally, the buffer will become empty and the profile generator becomes inactive.

Example of buffered instructions:

1-10 Mechanical system

1-10-1 Inertia ratio

The inertia ratio is a stability criterion. The higher the intertia of the load in relation to the intertia of the motor, the lower the gains you can set in your system before you reach oscillation, and the lower the performance you can reach.
With a ratio of 1:30 for small Servo Drivers and a ratio of 1:5 for big Servo Drivers you can reach the maximum dynamic of the motor-driver combination.

1-10-2 Rigidity

If a machine is more rigid and less elastic, you can set higher gains without vibration, and you can reach higher dynamic and lower Following Error.

1-10-3 Resonant frequency

A mechanical system has at least one resonant frequency. If you excite your mechanical system to the resonant frequency, it starts oscillating. For motion systems, it is best to have mechanical systems with a very high resonant frequency, that is, with low inertia and high rigidity.
The resonant frequency of the mechanical system is the limit for the gain settings.

1-11 Axis numbers

The CJ1W-MCH72 has a maximum of 31 non-virtual axes:

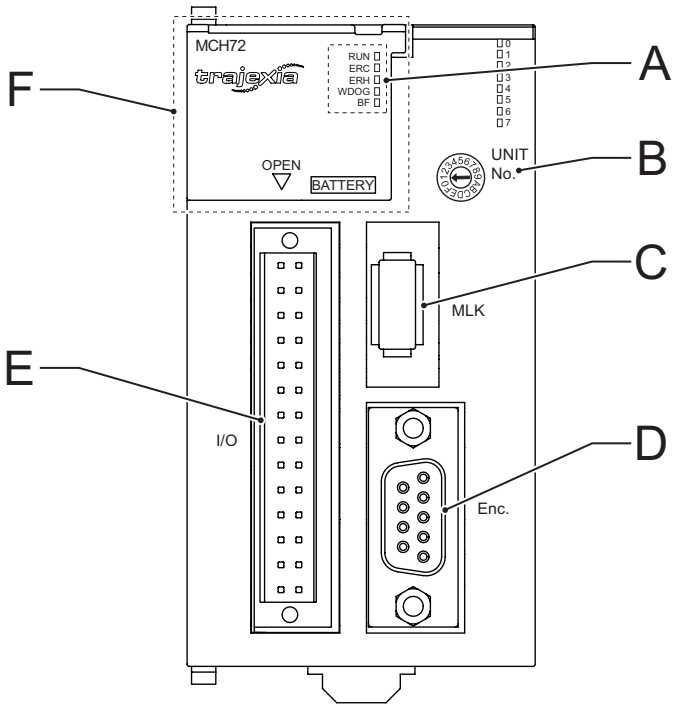
- 30 MECHATROLINK-II axes
- 1 Flexible axis (on the Encoder Interface)

MECHATROLINK-II slaves can have a station address that ranges from 41 hex to 5F hex. These station addresses correspond to axis numbers 0 to 29. The first non-assigned axis number, that is, the first axis number that is not assigned to a MECHATROLINK-II station address, is used for the flexible axis on the CJ1W-MCH72 encoder interface.

SECTION 2
Installation and wiring

2-1 Unit components

The CJ1W-MCH72 Motion Control Unit has the following components:

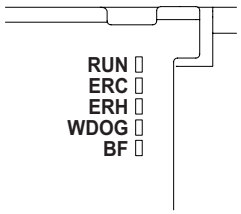


CJ1W-MCH72 Motion Control Unit

Label	Description
A	Status LED indicators
B	Unit number selector switch
C	MECHATROLINK-II connector
D	Encoder connector
E	I/O connector
F	Battery compartment

2-1-1 Status LED indicators

The CJ1W-MCH72 has 5 status LEDs. They indicate the operational mode and the status of the CJ1W-MCH72. The status LEDs are given in the table below.



Status LEDs

LED	Color	Status	Description
RUN	Green	OFF	• Startup test failed, unit not operational • Fatal error, operation stopped
		Flashing	Detection of MECHATROLINK-II slaves and assignment of axes in progress
		ON	Unit is ready to execute BASIC commands
ERC	Red	OFF	Unit is in normal operation
		Flashing during startup	Hardware error
		Flashing during execution	Low battery
		ON	Error log access error
ERH	Red	OFF	CPU in normal operation
		ON	Communication error with PLC CPU
WDOG	Green	OFF	Unit does not operate a Servo Driver
		ON	Unit operates a Servo Driver
BF	Red	OFF	Normal operation
		ON	MECHATROLINK-II bus fault

Also, the CJ1W-MCH72 has 8 general-purpose LEDs. The function of the general-purpose LEDs can be controlled with the **DISPLAY** system parameter. The table below lists the configuration for the LEDs and the **DISPLAY=n** command where **n** ranges from 0 to 7.

LED	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7
0	IN0	IN8	IN16	IN24	OUT0	OUT8	OUT16	OUT24
1	IN1	IN9	IN17	IN25	OUT1	OUT9	OUT17	OUT25
2	IN2	IN10	IN18	IN26	OUT2	OUT10	OUT18	OUT26
3	IN3	IN11	IN19	IN27	OUT3	OUT11	OUT19	OUT27
4	IN4	IN12	IN20	IN28	OUT4	OUT12	OUT20	OUT28
5	IN5	IN13	IN21	IN29	OUT5	OUT13	OUT21	OUT29
6	IN6	IN14	IN22	IN30	OUT6	OUT14	OUT22	OUT30
7	IN7	IN15	IN23	IN31	OUT7	OUT15	OUT23	OUT31

For example: if the command **DISPLAY=1** is executed, LED 5 reflects the activity of input IN13 (pin 16) of the 28-pin I/O connector.

2-1-2 Unit number selector switch

The unit number identifies each individual CPU bus unit when more than one CPU bus unit is connected to the same PLC. The unit number must be unique for each CPU bus unit. If the unit number is not unique, the PLC system cannot start correctly.

Unit number selector switch

The unit number can range from 0 hex to F hex.

2-1-2-1 Word allocations for CPU bus units

Words are automatically allocated in the CIO area of CJ-series PLC systems. The CJ1W-MCH72 uses these words to receive control data from the CPU and to notify the CPU of its status and the status of the communication.
The word addresses in the allocated areas depend on the unit number. Because the word addresses are hard-coded in user programs, a user program becomes invalid when the unit number is changed.
The table below gives the relation between the unit number and the allocated CIO area words.

Unit number	Allocated words	Unit number	Allocated words
0 hex (0)	CIO1500..CIO1524	8 hex (8)	CIO1700..CIO1724
1 hex (1)	CIO1525..CIO1549	9 hex (9)	CIO1725..CIO1749
2 hex (2)	CIO1550..CIO1574	A hex (10)	CIO1750..CIO1774
3 hex (3)	CIO1575..CIO1599	B hex (11)	CIO1775..CIO1799
4 hex (4)	CIO1600..CIO1624	C hex (12)	CIO1800..CIO1824
5 hex (5)	CIO1625..CIO1649	D hex (13)	CIO1825..CIO1849
6 hex (6)	CIO1650..CIO1674	E hex (14)	CIO1850..CIO1874
7 hex (7)	CIO1675..CIO1699	F hex (15)	CIO1875..CIO1899

2-1-3 Battery

The battery makes sure that the following RAM data is preserved when the power supply is off:

- User programs
- VR variables
- Table memory

If the battery is not installed, or the battery voltage is too low, the RAM data is lost when the power supply is off.

2-1-3-1 Battery lifetime

The maximum lifetime of a battery is 5 years when the ambient temperature is 25°C. The lifetime of the battery is shorter when the ambient temperature is higher. The lifetime of the battery is also shorter when there is no power supply to the unit for long periods.

2-1-3-2 Low-battery indicator

If a low-battery error occurs, the ERC LED flashes, the **BATTERY_LOW** parameter is **ON**, and bit 3 of status word *n*+2 is set (see section 3-3-1-2). If this occurs, perform the following steps:

1 Check if the battery is installed correctly.
2 If the battery is installed correctly, replace it.

When a low-battery error occurs, the battery can continue to function for 5 days if the ambient temperature is 25°C and if power is supplied to the PLC system at least one time per day. If the power is not turned off until the battery is replaced, the battery failure and the resulting loss of RAM data can be delayed.
If the ambient temperature is more than 25°C, the battery can only continue for less than 5 days. If the ambient temperature is 40°C, the battery can continue for 4 days. If the ambient temperature is 55°C, the battery can continue for 2 days.

2-1-3-3 Replacing the battery

When replacing the battery, make sure that the following is adhered to:

- Use the CJ1W-BAT01 as a replacement battery.
- Do not use a replacement battery that is older than 2 years. The production date of the battery is shown on the label, in the format *yy-mm*, where *yy* is the year and *mm* is the month.

Production date of the battery: March 2008.

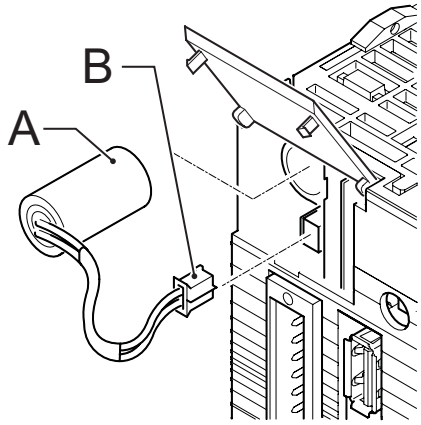
- Make sure that the sensitive internal components of the CJ1W-MCH72 cannot be damaged by static electricity when the battery is replaced. This can be done in two ways:
  1 Turn the power off. (This is the recommended way.)
  2 Leave the power on, and touch a grounded piece of metal to discharge static electricity before replacing the battery.
- Make sure that an experienced technician is in charge when the battery is replaced. This is required by UL standards.

To replace the battery, perform the following steps:

1 Make sure that the power to the unit is on for at least 5 minutes.
2 Turn the power to the unit off.

Note You must perform the following steps within 5 minutes. If you do not perform the steps within 5 minutes, the RAM can be deleted.

3 Open the battery compartment.
4 Remove the old battery (A) from the battery compartment.



Battery (A) and battery connector (B)

5 Remove the battery connector (B) of the old battery.
6 Connect the battery connector (B) of the new battery.
7 Install the new battery (A) in the battery compartment.
8 Close the battery compartment.

When the new battery is installed, the low-battery error is cleared.

⚠WARNING These actions can cause the battery to leak, burn or rupture, which can lead to fire, injury, loss of property and death.
- Do not short-circuit the battery terminals.
- Do not charge the battery.
- Do not disassemble the battery.
- Do not heat the battery or set fire to the battery.

⚠WARNING Do not use a battery that is dropped on the floor or that is subjected to a shock. The battery can leak.

2-1-4 I/O connector

The I/O connector is a 28-pin connector used to wire to an external I/O. All I/Os are general-purpose I/Os. Functions like limit inputs and origin proximity inputs can be allocated.
The recommended connector is the Weidmuller B2L 3.5/28 LH connector.

2-1-5 MECHATROLINK-II connector

The MECHATROLINK-II connector is used to connect the CJ1W-MCH72 unit to a MECHATROLINK-II network.

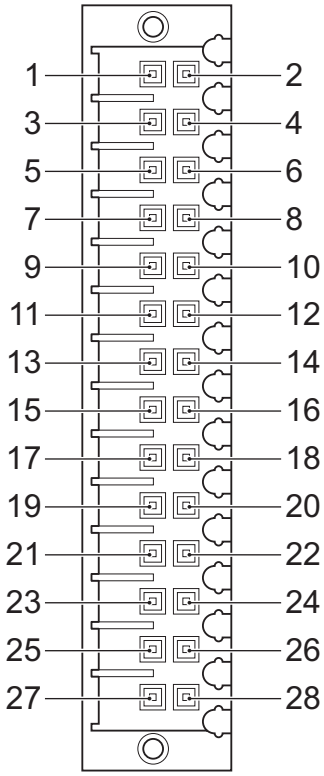
2-1-6 Encoder connector

The encoder connector of the MCH72 supports different types of encoders. It can also act as a reference output. It supports the following:
- Incremental encoder input, for line-driver type encoders
- Two different absolute encoder standards: SSI and EnDat
- Incremental encoder output, to simulate a line-driver type encoder
- Stepper output, to control stepper drivers

2-2 Wiring

2-2-1 I/O connector

The I/O connector is a 28-pin connector. Input 0 and input 1 can also be used as registration inputs.



I/O connector pins

The connections of the pins are given in the table below.

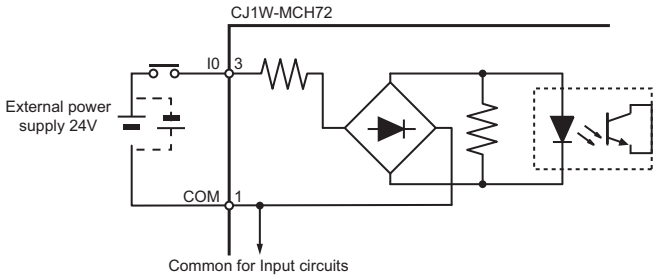
Pin	Connection	Pin	Connection
1	Input common	2	Input common
3	Registration input 0	4	Registration input 1
5	Input 2	6	Input 3
7	Input 4	8	Input 5
9	Input 6	10	Input 7
11	Input 8	12	Input 9
13	Input 10	14	Input 11
15	Input 12	16	Input 13
17	Input 14	18	Input 15
19	Output 8 (PSWITCH)	20	Output 9
21	Output 10	22	Output 11
23	Output 12	24	Output 13
25	Output 14	26	Output 15

Pin	Connection	Pin	Connection
27	0 V Output common	28	24 V Power supply input for the outputs

2-2-1-1 Specifications

The table below shows the digital input specifications of input 0 to input 3 for the I/O.

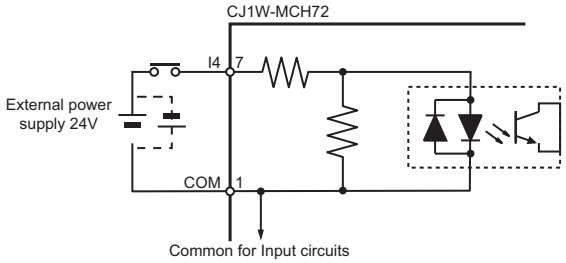
Item	Specification
Type	NPN/PNP
Maximum voltage	24 VDC +10%
Input current	8 mA at 24 VDC
ON voltage	17 VDC min.
OFF voltage	5 VDC max.
Maximum response time	1250 μs if the servo period equals 0.5 ms or 1.0 ms 2500 μs if the servo period equals 2.0 ms
Registration response time (Inputs 0 and 1)	1 μs typical
Ambient operating temperature	0°C to 55°C When all digital inputs are on: max. 45°C When max. 8 digital inputs are on: max. 55°C



Circuit configuration for input 0 to input 3

The table below shows the digital input specifications of input 4 to input 15 for the I/O.

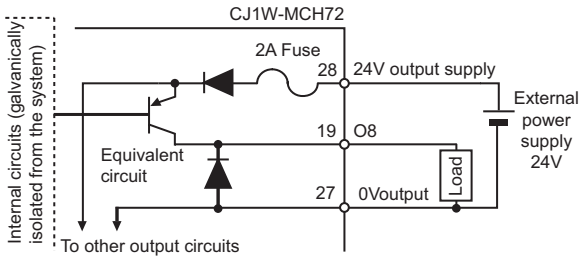
Item	Specification
Type	NPN/PNP
Maximum voltage	24 VDC +10%
Input current	3.2 mA at 24 VDC
ON voltage	12 VDC min.
OFF voltage	5 VDC max.
Maximum response time	1250 μs if the servo period equals 0.5 ms or 1.0 ms 2500 μs if the servo period equals 2.0 ms
Ambient operating temperature	0°C to 55°C When all digital inputs are on: max. 45°C When max. 8 digital inputs are on: max. 55°C



Circuit configuration for input 4 to input 15

The table below shows the digital output specifications of output 8 to output 15 for the I/O.

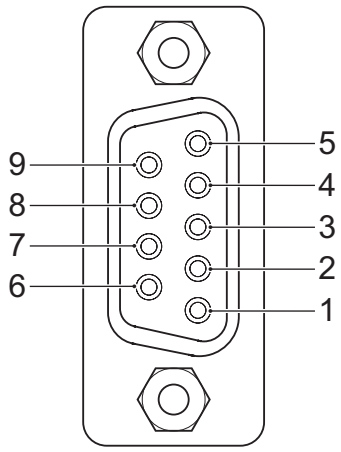
Item	Specification
Type	PNP
Maximum voltage	24 VDC +10%
Current capacity	100 mA for each output 800 mA in total for the group of 8 outputs
Protection	Over current Over temperature 2 A fuse on common
Maximum response time	250 μs for ON, 350 μs for OFF, if the servo period equals 0.5 ms or 1.0 ms 500 μs for ON, 600 μs for OFF, if the servo period equals 2.0 ms 150 μs when the **HW_PSWITCH** is used



Circuit configuration for output 4 to output 15

2-2-2 Encoder connector

The connections of the pins are given in the table below.



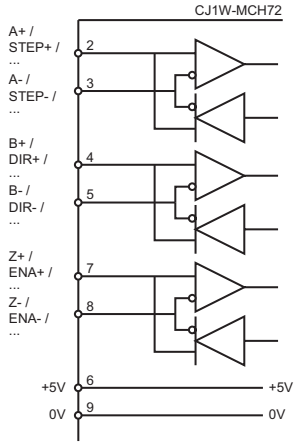
Encoder connector pins

Pin	Incremental encoder input	Incremental encoder output	Stepper output	SSI, EnDat
1	NC			
2	A+	A+	Step+	Clock+
3	A-	A-	Step-	Clock-
4	B+	B+	Dir+	
5	B-	B-	Dir-	
6	5 V Encoder power supply			
7	Z+	Enable+	Enable+	Data+
8	Z-	Enable-	Enable-	Data-
9	0 V Encoder ground			
Shell	FG			

2-2-2-1 Specifications

The table below shows the specifications.

Item	Specification
Signal level	EIA RS-422A Standards
Input impedance	48 kΩ min.
Current capacity	20 mA
Termination	None
Maximum response time registration	0.5 μs



Circuit configuration for the encoder interface

2-2-3 Incremental encoder

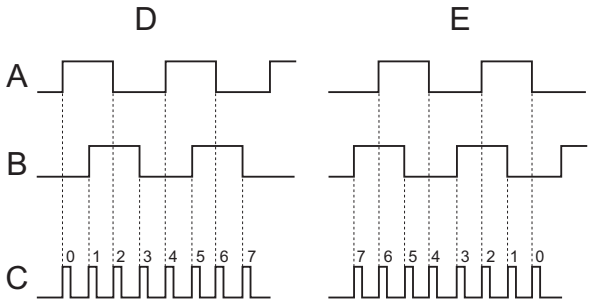
An incremental encoder has the following phase definition:

- An advanced phase A for forward rotation.
- An advanced phase B for reverse rotation.

By monitoring the relative phase of the 2 signals, you can easily detect the rotation direction. If signal A leads signal B, the movement is clockwise and the counter increments. If channel B leads channel A, the movement is counterclockwise and the counter decrements.
Most rotary encodes also provide an additional Z marker. This Z marker is a reference pulse within each revolution. With these 3 signals, you can determine the direction, the speed and the relative position.

2-2-3-1 Encoder input

The pulse ratio of the CJ1W-MCH72 is 1: every encoder edge (i.e., a pulse edge for either phase A or B) is equal to one internal count.



Encoder edges

The figure shows phase A (A), phase B (B) and the number of counts (C) for forward or clockwise rotation (D) and reverse or counterclockwise rotation (E). The signals A, B and Z appear physically as A+ and A-, B+ and B- and Z+ and Z-. They appear as differential signals on twisted-pair wire inputs. This makes sure that common mode noise is rejected. When you use an encoder from other manufacturers, check the encoder specification for the phase

advancement carefully. If the phase definition is different from the phase definition of the standard OMRON equipment, reverse the B-phase wiring between the CJ1W-MCH72 and the encoder.

Note The encoder interface of the CJ1W-MCH72 does not have termination inside. In case of long distances or disturbed communication, add external termination to the interface.

2-2-3-2 Registration

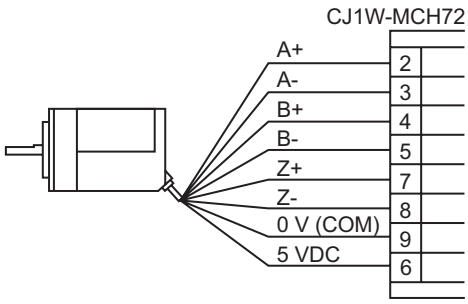
When using the incremental encoder interface of the CJ1W-MCH72, the CJ1W-MCH72 can capture the position of the Flexible Axis in a register when an event occurs. The event is called the print registration input. On the rising or falling edge of an input signal (either the Z marker or one of the first 2 digital inputs), the CJ1W-MCH72 captures the position of the axis in the hardware. You can use this position to correct possible errors between the actual position and the desired position.
To set up the print registration, you can use the **REGIST** command. The position is captured in the hardware, which means that there is no software overhead. Therefore, you do not have to deal with timing issues.
For more information on how to use the registration inputs, refer to the **REGIST** command in section 4-2-198.

2-2-3-3 Hardware PSWITCH

The MCH72 has one output (output 8) that can be used as a hardware position switch. This output goes on when the measured position of the Flexible Axis is reached. It goes off when another measured position is reached.
The output is driven by hardware only. This means that the response times do not have software delays. For more information on using the position switch, refer to section 4-2-129 on the **HW_PSWITCH** command.

2-2-3-4 Connection example

The table below and the figure give an example of the OMRON E6B2-CWZ1Z encoder connected to the CJ1W-MCH72.

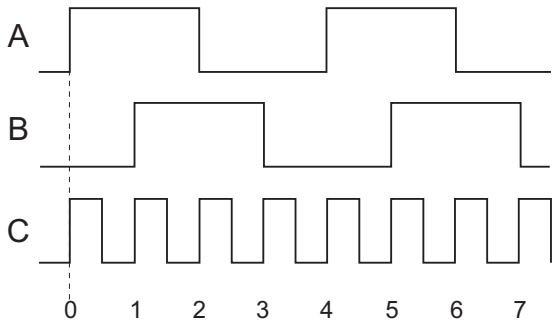


Encoder input connection

E6B2-CWZ1Z encoder		CJ1W-MCH72 encoder interface	
Signal	**Wire color**	**Pin**	**Signal**
A+	Black	2	A+
A-	Black/red	3	A-
B+	White	4	B+
B-	White/red	5	B-

E6B2-CWZ1Z encoder		CJ1W-MCH72 encoder interface	
Signal	**Wire color**	**Pin**	**Signal**
Z+	Orange	7	Z+
Z-	Orange/red	8	Z-
0 V (COM)	Blue	9	0 V Encoder ground
5 VDC	Brown	6	5 V

2-2-3-5 Encoder output



Encoder edges

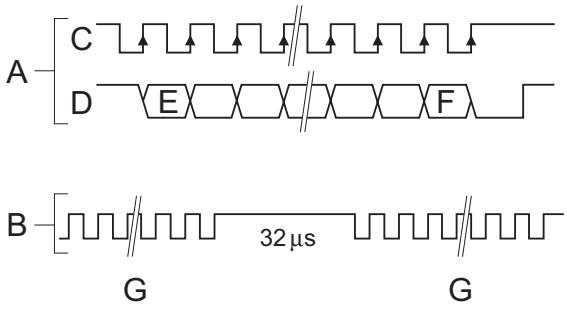
The CJ1W-MCH72 can generate encoder-type pulses. For each internal count (C), the CJ1W-MCH72 produces one encoder edge for phase A (A) or phase B (B).

2-2-4 Absolute encoder

2-2-4-1 SSI

SSI (Synchronous Serial Interface) is a digital system for transferring data in serial form. SSI is the most widely used serial interface between absolute sensors and controllers. SSI uses a pulse train from the controller to clock out the data from the sensor. The SSI interface of the CJ1W-MCH72 accepts absolute values from an encoder if the data is in Gray Code format or in binary format and if the resolution is 24 bits or less. The number of bits, and therefore the number of clock pulses sent to the encoder in each frame, is programmable. You set this number with the BASIC command **ENCODER_BITS = n**.

When you have initialized the CJ1W-MCH72 with the **ENCODER_BITS** command, the CJ1W-MCH72 continuously sends clock pulses to the encoder. These clock pulses are sent in frames of *n*+2 pulses, where *n* is the bit count set. The clock rate is fixed at 200 kHz. The clock interval between frames is 32 µs. The resulting maximum cable length between the controller and the sensor is 200 m.



SSI pulses

The labels in the figure are:
A Timing diagram
B Clock sequence
C Clock
D Data
E MSB (Most Significant Bit)
F LSB (Least Significant Bit)
G Clock frame

When the data is clocked into the CJ1W-MCH72, the position value is interpreted. With this position value, it produces a value for **MPOS** and a position error that is used to close the control loop.
The connections for SSI are:

Pin	Signal
2	Clock+
3	Clock-
6	5 V
7	Data+
8	Data-
9	0 V

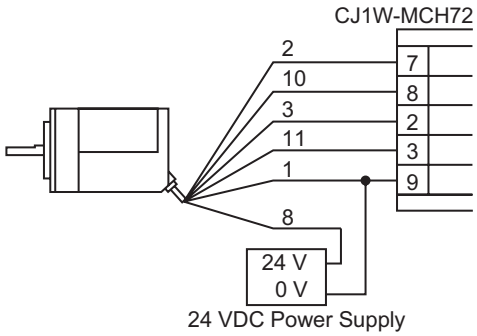
Note The CJ1W-MCH72 encoder interface does not have a termination inside. In case of long distances or disturbed communication, add an external termination to the interface.

The table below and the figure give an example of how to connect the Stegmann ATM 60-A encoder to the CJ1W-MCH72.

Encoder			CJ1W-MCH72 encoder interface	
Pin	Signal	Wire color	Pin	Signal
2	Data+	White	7	Data+
10	Data-	Brown	8	Data-
3	Clock+	Yellow	2	Clock+
11	Clock-	Lilac	3	Clock-

Encoder			CJ1W-MCH72 encoder interface	
Pin	**Signal**	**Wire color**	**Pin**	**Signal**
1	GND	Blue	9	0 V Encoder ground
8	Us	Red	See footnote[1]	

1. Use an external power supply



Stegmann ATM 60-A connection

2-2-4-2 EnDat

You can configure the CJ1W-MCH72 to interface directly to EnDat absolute encoders. EnDat absolute encoders respond on a dedicated Clock and Data 1 MHz RS485 serial interface when their position is requested by the controller. When you set the encoder to the relevant encoder mode, the axis transmits an information request to the encoder on a fixed 250 µs cycle.
The connections for EnDat are:

Pin	Signal
2	Clock+
3	Clock-
6	5 V
7	Data+
8	Data-
9	0 V

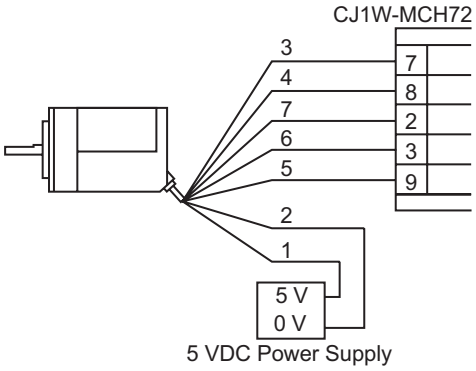
Note The CJ1W-MCH72 encoder interface does not have a termination inside. In case of long distances or disturbed communication, add an external termination to the interface.

The table below and the figure give an example of the connection of the Heidenhain ROC 425 2048 5XS08-C4 encoder to the CJ1W-MCH72.

Encoder			CJ1W-MCH72 encoder interface	
Pin	**Signal**	**Wire color**	**Pin**	**Signal**
3	Data	Grey	7	Data+
4	/Data	Pink	8	Data-
7	Clock	Violet	2	Clock+

Encoder			CJ1W-MCH72 encoder interface	
Pin	**Signal**	**Wire color**	**Pin**	**Signal**
6	/Clock	Yellow	3	Clock-
5	GND	White/green	9	0 V Encoder ground
2	0 V	White	See footnote[1]	
1	Up	Blue		

1. Use an external power supply

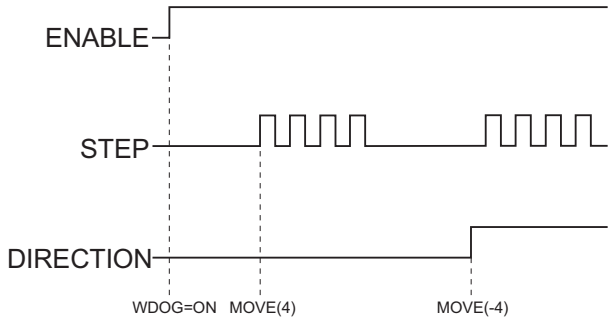


Heidenhain ROC 425 2048 5XS08-C4 connection

2-2-4-3 Stepper

The CJ1W-MCH72 can generate pulses to drive an external stepper motor amplifier. You can use single step, half step and microstepping drivers with this interface. The applicable signals are:

- Enable
- Step
- Direction



The applicable signals when a **MOVE** operation is performed

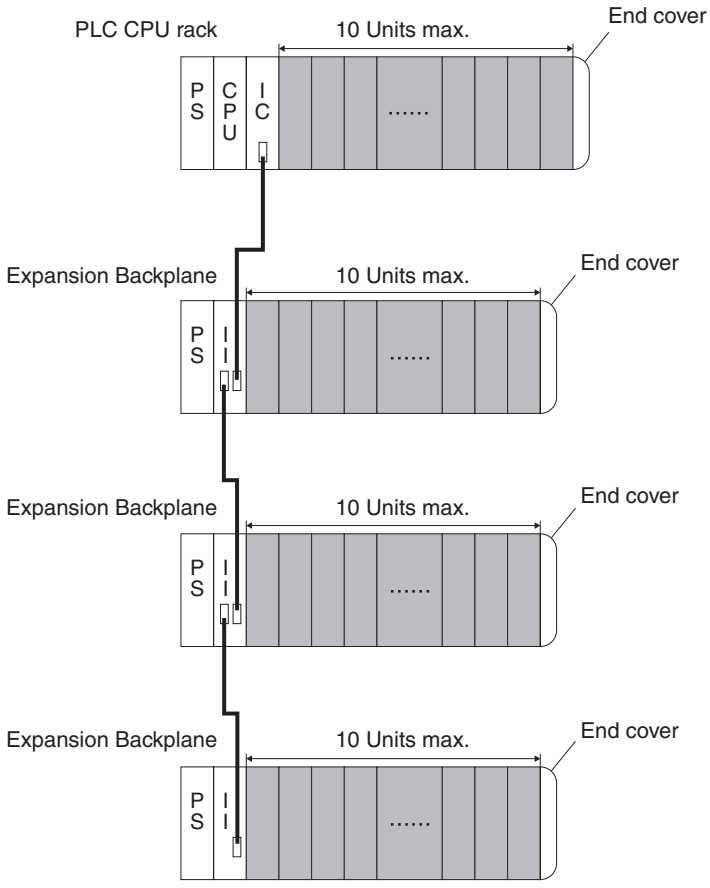
2-3 Installation

2-3-1 Hardware installation

⚠ **Caution** Obey the following precautions when you install the CJ1W-MCH72 in a PLC system:

- Turn off the power supply to the PLC before the installation or connection of the CJ1W-MCH72.
- Use separate conduits or ducts for the I/O lines. This prevents noise from high-tension lines or power lines.
- Do not remove the label on top of the CJ1W-MCH72 during the installation and wiring. The label makes sure that no foreign matter can enter the unit.
- Remove the label on top of the CJ1W-MCH72 after the installation and wiring of the unit. This makes sure that the unit cannot become overheated.

The CJ1W-MCH72 can be installed in any slot in a CJ-series CPU rack or in a CJ-series expansion CPU rack. The CJ-series PLC supports up to 4 expansion CPU racks.
Up to 16 CPU bus units can be connected to one PLC. Also, the maximum number of CJ1W-MCH72 units that can be connected to one PLC is 16.

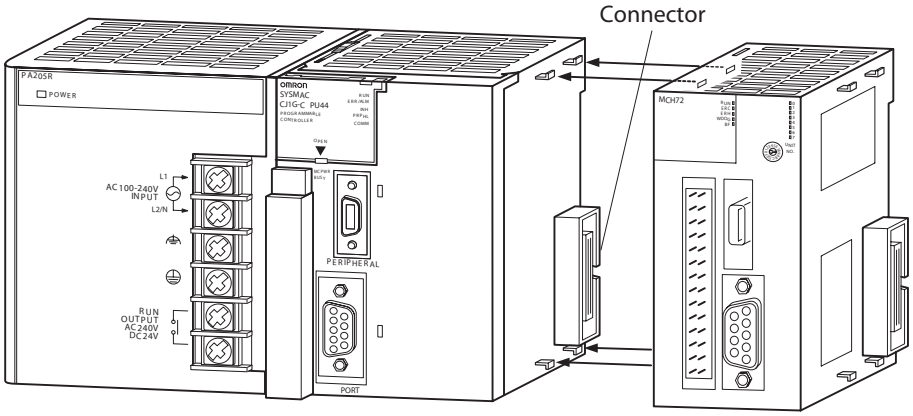


Expansion racks

Note The maximum current consumption of the CJ1W-MCH72 is 680 mA. Make sure that the total current consumption of all units connected to the same CPU backplane or expansion backplane is not greater than the output capacity of the Power Supply Unit.

To connect the CJ1W-MCH72 to the PLC, perform the following steps:

1 Align the connectors of the units.



CJ1W-MCH72 hardware installation

2 Move the yellow sliders at the top and bottom of the unit to the front.
3 Attach the CJ1W-MCH72 to the PLC.
4 Push the yellow sliders at the top and bottom of the unit to the rear to lock them.

Note If the sliders are not properly locked, it is possible that the unit does not operate correctly.

Note Communication errors can occur in the Encoder Interface when Contact Output Units are installed close to the CJ1W-MCH72. This is caused by noise generated by the Contact Outputs.
If Contact Output Units and a CJ1W-MCH72 are installed on the same rack and communication errors occur, do one of the following:

- Install the Contact Output Units at maximum distance from the CJ1W-MCH72.
- Use surge absorbers for the Contact Outputs.

2-3-2 Setup

After the installation of the unit in a PLC system, the following initial setup procedure must be executed:

- A unique unit number must be set. Refer to section 2-3-2-1.
- An I/O table must be created in the PLC, to register the unit on the PLC CPU. Refer to section 2-3-2-2.

This initial setup procedure makes sure that the unit can start up properly and can be configured for operation.

2-3-2-1 Setting the unit number

To set the unit number, perform the following steps:

1 Turn off the power supply to the PLC system.
2 Use a small screwdriver to set the unit number selector switch to the new unit number.

⚠ **Caution** Do not damage the unit number selector switch.

Note The factory setting of the unit number selector switch is 0.

3 Turn on the power supply to the PLC system.

Note The unit reads the unit number during the initialization after a power up. It does not read the unit number after a software reset. Thus, the power must always be turned off before the unit number is set.

Note If the unit number is set for the first time, or if the unit number is changed, an I/O table must be created for the PLC system.

2-3-2-2 Creating an I/O table

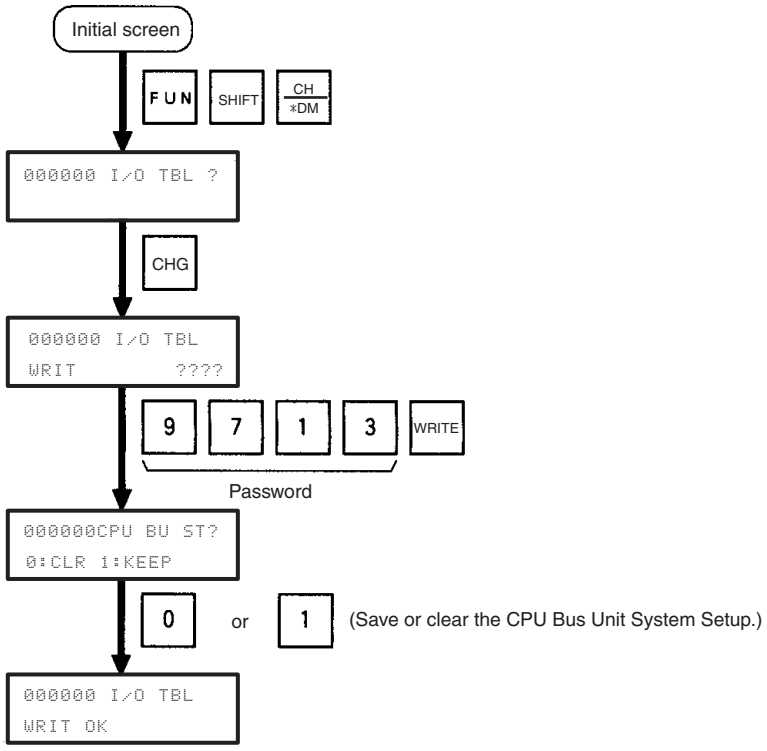
The I/O table identifies the units connected to the PLC and allocates I/O to these units. The I/O table is stored in the PLC CPU. It is loaded at start-up.
If the configuration of a unit connected to the PLC is changed, the I/O table must be created again to register the units to the CPU.
To create the I/O table, connect a programming device (such as a programming console or the CX-Programmer software) to the PLC. The following programming console can be used:

- Model number: C2090H-PRO27-E
- Key sheet (required): CS1W-KS001-E
- Recommended cable (required):
  - CS1W-CN224 (2 m)
  - CS1W-CN624 (6 m)

For CX-Programmer, refer to the CX-Programmer User Manual.

Below is given the procedure to create an I/O table with a programming console.

1 Attach a key sheet to the programming console.
2 Connect the programming console to the peripheral port of the CPU. Do not connect it to the RS-232C port.
3 Follow the steps on the programming console given in the figure.



Create I/O table

2-3-3 Connecting MECHATROLINK-II slaves

Note The Trajexia system supports 3 kinds of MECHATROLINK-II slaves: Servo Drivers, Inverters and I/Os.
The CJ1W-MCH72 only supports 2 kinds of MECHATROLINK-II slaves: Servo Drivers and Inverters. It does not support I/Os.

To connect MECHATROLINK-II slaves, use a MECHATROLINK-II cable for W-series with ring core and USB connector on both ends. These cables are not included.
The table below lists the MECHATROLINK-II cables.

OMRON model	Yaskawa model	Cable length
FNY-W6003-A5	JEPMC-W6003-A5	0.5 m
FNY-W6003-01	JEPMC-W6003-01	1 m
FNY-W6003-03	JEPMC-W6003-03	3 m
FNY-W6003-05	JEPMC-W6003-05	5 m
FNY-W6003-10	JEPMC-W6003-10	10 m
FNY-W6003-20	JEPMC-W6003-20	20 m
FNY-W6003-30	JEPMC-W6003-30	30 m

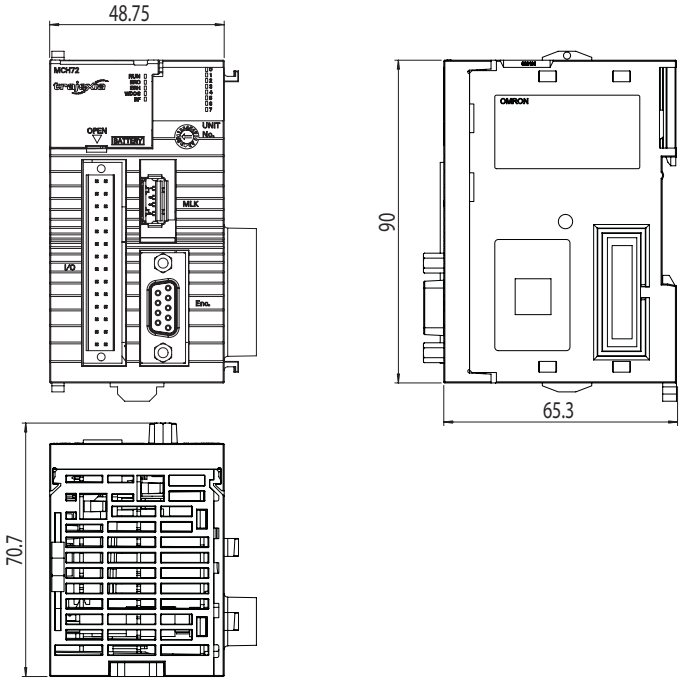
To terminate the MECHATROLINK-II slaves, connect an OMRON FNY-W6022 terminator or a Yaskawa JEPMC-W6022 terminator to the MECHATROLINK-II connector of the last MECHATROLINK-II slave.
When the MECHATROLINK-II slaves are connected up to 16 nodes (within 30 m) or up to 15 nodes (within 50 m), no repeater unit is required. For more nodes or longer distances, a repeater unit is required. You can use the OMRON FNY-REP2000 repeater or the Yaskawa JEPMC-REP2000 repeater.

Note For more details, refer to the Yaskawa Sigma-II Series SGDH MECHATROLINK-II Application Module User Manual for model JUSP-NS115 (manual number SIEPC71080001.

MECHATROLINK-II slaves can have a station address that ranges from 41 hex to 5F hex. These station addresses correspond to axis numbers 0 to 29.

2-4 Specifications

2-4-1 Unit dimensions



Dimensions of the CJ1W-MCH72

2-4-2 Unit specifications

Item	Specification
Ambient operating temperature	0°C to 55°C When all digital inputs are on: max. 45°C When max. 8 digital inputs are on: max. 55°C
Ambient storage temperature	−20°C to 70°C
Ambient operating humidity	10% to 90%RH
Ambient storage humidity	90% max. (without condensation)
Atmosphere	No corrosive gases

Item	Specification
Vibration resistance	10 to 57 Hz (0.075 mm amplitude) 57 to 100 Hz (Acceleration: 9,8 m/s^2, in X, Y and Z directions for 80 minutes)
Shock resistance	143 m/s^2, 3 times each, X, Y and Z directions
Insulation resistance	20 MΩ
Dielectric strength	500 V
Protective structure	IP20
International standards	CE, EN61131-2, RoHS compliant
Power supply	5 V (Supplied via PLC bus)
Current consumption	450 mA max. without external load 680 mA max. with maximum external load
Maximum current ratings Encoder Interface 5 V output	Incremental: 150 mA SSI: 100 mA EnDat: 80 mA Stepper: 80 mA
Weight	180 gr

2-4-3 System specifications

Item	Specification
Number of axes	32
Number of inverters	Max. 8
Cycle time	Selectable: 0.5 ms, 1 ms, 2 ms, 4 ms
Programming language	BASIC-like motion language
Multi-tasking	Max. 14 tasks running simultaneously
Built-in digital I/O	16 Inputs, 2 with registration functionality 8 Outputs, 1 with hardware position switch functionality
Measurement units	User-definable
Available memory for user programs	756 - 1006 KB[1]
Data storage capacity	RAM memory: 1024 KB minus the size of the user programs VR memory: 4 KB
Saving program data (unit)	SRAM with battery backup and Flash-ROM
Saving program data (PC)	Trajexia Studio manages a backup on the hard disk of the PC
Firmware update	Via Trajexia Studio

1. The available memory for user programs depends on the size of the Table memory, which is minimum 0 KB and maximum 250 KB.

2-4-4 MECHATROLINK-II specifications

Item	Specification
Electrical characteristics	Conforms to the MECHATROLINK standard
Communication ports	1 MECHATROLINK-II master port
Transmission speed	10 Mbps
Communication cycle	0.5 ms, 1 ms, 2 ms, 4 ms
Slave types	Servo Drivers Frequency Inverters
Number of slaves per master (Cycle time)	Max. 30 slaves (4 ms) Max. 16 slaves (2 ms) Max. 8 slaves (1 ms) Max. 4 slaves (0.5 ms)
Transmission distance	Max. 50 m

2-4-5 Encoder interface specifications

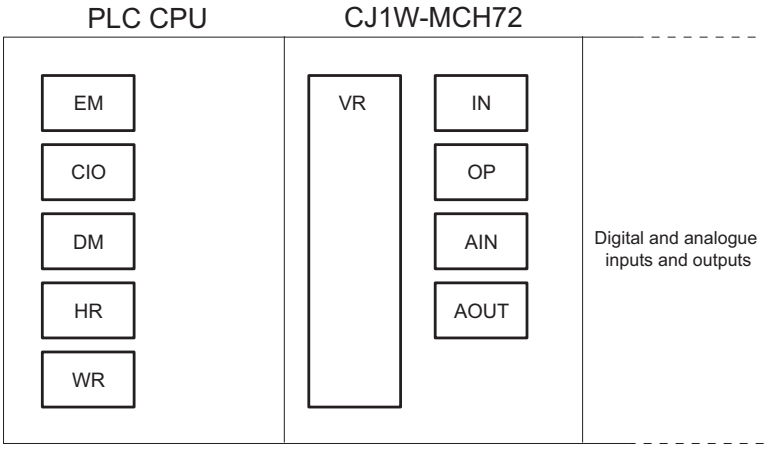
Item	Specification
Number of axes	1
Electrical characteristics	EIA RS-422-A Standards (line-driver)
Control method	Pulse Train output (open loop only)
Encoder position/speed feedback	Incremental and absolute
Absolute encoder standards supported	SSI 200 kHz EnDat 1 MHz
Encoder input maximum edge rate	6 M-edges/s
Encoder/pulse output maximum edge rate	2 M-edges/s
Maximum cable length	SSI 100 m max. EnDat 40 m max. Encoder input 100 m max. Encoder/stepper output 100 m max.

SECTION 3
Data exchange

3-1 Introduction

The CJ1W-MCH72 can exchange data with memory areas in the PLC. This enables the CJ1W-MCH72 to use the inputs and outputs connected to the PLC. Also, programs in the CJ1W-MCH72 and PLC programs can exchange control and status data.
Because the CJ1W-MCH72 can only access the input and output data via the PLC, the data exchange requires an extra PLC cycle.
This section describes the issues related to cyclic data exchange.

3-2 Memory areas



Data exchange memory areas

The PLC CPU uses the following memory areas for data exchange with the CJ1W-MCH72:
- EM (Expanded Memory)
- CIO (Common I/O memory)
- DM (Data Memory)
- WR memory
- HR memory

The CJ1W-MCH72 uses the following memory areas to exchange data with the PLC CPU:
- VR memory
- IN array (for digital inputs)
- OP array (for digital outputs)
- AIN array (for analogue inputs)
- AOUT array (for analogue outputs)
- Axis Status array (see section 3-3-2-1)

The mapping of memory areas for cyclic data exchange in the PLC CPU to memory areas in the CJ1W-MCH72 can be freely configured. This can be done in the PLC program or in the CJ1W-MCH72. It is recommended to configure the memory mapping either in the startup program of the PLC or in the startup program of the CJ1W-MCH72.

It is possible to configure the mapping of memory areas both in the PLC program and in the CJ1W-MCH72. This is not practical, because the last configuration overwrites the first.

Note Data exchange with the Table memory of the CJ1W-MCH72 is not possible. However, with the FINS Write command you can write the Table memory of the CJ1W-MCH72.

3-2-1 Configuration of memory areas in the PLC program

To configure the memory areas for cyclic data exchange in the PLC, you must use the FINS Parameter Area Write command. For more information on this FINS command, refer to section 3-4-4.

3-2-2 Configuration of memory areas in the CJ1W-MCH72

To configure the memory areas for cyclic data exchange in the CJ1W-MCH72, you must use the **PLC_EXCHANGE** BASIC command. For more information on this BASIC command, refer to section 4-2-182.

3-3 Data

Two types of data are exchanged during a data exchange:
- Control and status data
- Configurable data

3-3-1 Control and status data

The CJ1W-MCH72 has 13 control and status words.
The control and status data is in the PLC CIO memory at word location *n*, where *n* is equal to 1500 + 25 × *unit_number*.

3-3-1-1 Control data

The PLC program can directly control the CJ1W-MCH72 with control word *n*. It can execute the following functions:

Function	Action
Disable all program execution (Execute BASIC command **HALT**)	Set bit 0 to 0
Enable all program execution (On the first edge after reset or power-on, the startup program is executed)	Set bit 0 to 1
Set the axes watchdog off	Set bit 1 to 0
Start deceleration (Execute BASIC commands **HALT : RAPIDSTOP**)	Set bit 2 to 1

Note
- Only if bit 1 has value 1 the CJ1W-MCH72 can react to the axes watchdog.
- Setting bit 2 to 0 has no function.
- Bits 3..15 of word *n* are ignored.

3-3-1-2 Status data

Status words *n*+1 and *n*+2 return the status of the CJ1W-MCH72. The table below lists the layout of these status words.

Word	Bit	Description	Value	Function
n+1	0	Unit operational	0	Unit not operational
			1	Unit operational
	1	Watchdog on	0	Axes watchdog off
			1	Axes watchdog on
	2..15	N/A	Always 0	
n+2	0	MECHATROLINK-II error	0	No MECHATROLINK-II error
			1	Error on MECHATROLINK-II bus
	1	Axes error	0	No axes error
			1	Axes error (See control and status words n+7 and n+8)
	2	BASIC error	0	No BASIC error
			1	BASIC error for a running process
	3	Battery error	0	Battery OK
			1	Battery low or empty
	4..15	N/A	Always 0	

Status words *n*+3..*n*+12 return the status of the axes and processes of the CJ1W-MCH72.

Word	Bit	Description
n+3	0..15	Servo On flags for axes 0..15
n+4	0..15	Servo On flags for axes 16..31
n+5	0..15	Axis Enable flags for axes 0..15
n+6	0..15	Axis Enable flags for axes 16..31
n+7	0..15	Axis Error flags for axes 0..15
n+8	0..15	Axis Error flags for axes 16..31
n+9	0..15	Axis In Commissioning Mode flags for axes 0..15
n+10	0..15	Axis In Commissioning Mode flags for axes 16..31
n+11	0..13	Process Running flags for processes 1..14
	14,15	Always 0
n+12	0..13	Process Error flags for processes 1..14
	14,15	Always 0

3-3-2 Configurable data

The amount of configurable data that is exchanged between the PLC CPU and the CJ1W-MCH72 each PLC cycle is 8 blocks from the PLC CPU to the CJ1W-MCH72 and 8 blocks vice versa. Thus, 16 blocks of data are exchanged in one PLC cycle. A block is a continuous memory area or array area. The total size of all 16 blocks must be less than or equal to 2000 words. The configurable data can be exchanged between the VR, IN, OP, AIN, AOUT and Axis Status memory areas of the CJ1W-MCH72 and the CIO, DM and EM memory areas of the PLC CPU.

Because the PLC CPU and the CJ1W-MCH72 use different numeric formats, the data that is exchanged must be cast. The table below lists the casting of numeric data per memory area.

PLC data format	CJ1W-MCH72	
	Memory area	Data format
32-bit IEEE float	VR	floating point
16-bit word	VR	floating point
	IN	bit array
	OP	bit array
	AIN	floating point
	AOUT	floating point

3-3-2-1 Axis Status array

The Axis Status array is a special array that exists of 4 fields. The table below lists the Axis Status fields and the corresponding PLC data type.

Axis Status field	Description	PLC data type
Status	BASIC command **AXISSTATUS**	16-bit word
Position	BASIC command **MPOS**	32-bit integer
Monitor	BASIC command **DRIVE_MONITOR**	16-bit word
Drive status	Status of the drive	16-bit word

3-4 FINS commands

FINS (Factory Intelligent Network Services) commands are message service communications commands developed by OMRON for Factory Automation control devices. They do not depend on a particular transmission path, and can be used to:

- Read from and write to the PLC memory or the CJ1W-MCH72 memory.
- Control various operations.

The FINS communications can be issued from a PLC CPU or a host computer, and they can also be sent to any of these. The specific commands that can be sent depend on the destination.

A FINS command is defined by its command code and its response code. A command code is a 2-byte hexadecimal code. FINS commands always begin with a 2-byte command code. The required parameters come after the command code.
The response code is a 2-byte hexadecimal code that indicates the results of the command execution. The first byte provides the main response code (MRES), which classifies the results. The table below lists the MRES codes. The second byte provides the sub-response code (SRES), which contains details about the results.

MRES	Execution result
00	Normal completion
01	Master unit error
02	Slave device error

MRES	Execution result
04	Service not supported
10	Command format error
11	Parameter error
20	Read not possible
22	Status error

Note If the FINS command was not completed normally, the 2-byte response code is not equal to 0000. Also, the response frame does not contain any additional data. Data can only be returned - depending on the command - when the response code is 0000.

The CJ1W-MCH72 supports the following FINS commands:

- Read (0101)
- Write (0102)
- Parameter Area Read (0201)
- Parameter Area Write (0202)
- Run (0401)
- Stop (0402)

For more information on FINS, refer to the Communication Commands Reference manual (W342-E1).

3-4-1 Read (0101)

The FINS Read command has this format:

01	01	..	..	..	00	..	..
command_code		**var_type**	**start_address**		**fixed**	**element_count**	

The parameters can have the following values:

Parameter	Values (hex)
command_code	01 01
var_type	• 82 (Table memory in 16-bit integer format) • C2 (Table memory in 32-bit IEEE floating-point format) • B0 (VR memory in 16-bit integer format) • F0 (VR memory in 32-bit IEEE floating-point format)
start_address	0 ≤ **start_address** ≤ memory size − 1 ≤ FFFF
element_count	1 ≤ **element_count** ≤ memory size − **start_address**

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
var_type invalid	1101	No area type
start_address invalid	1103	Address range designation error
Number of elements invalid	1104	Address out of range

If **var_type** is 82 or B0, and the response code is 0000, the CJ1W-MCH72 responds with:

01 01	00 00			
command_code	**response_code**	**word_1**	**word_2**	**...**

If **var_type** is C2 or F0, and the response code is 0000, the CJ1W-MCH72 responds with:

01 01	00 00		
command_code	**response_code**	**dword_1**	**...**

Note The returned words and dwords are in big-endian format.

3-4-2 Write (0102)

The FINS Write command has these formats:

- If **var_type** is 82 or B0:

01 02	..	.. ..	00	.. ..	.. ..	.. ..
command_code	**var_ type**	**start_ address**	**fixed**	**total_words**	**word 1**	**..**

- If **var_type** is C2 or F0:

01 02	..	.. ..	00	.. ..	.. ..	.. ..	..
command_code	**var_type**	**start_ address**	**fixed**	**total_ dwords**	**dword 1**		**..**

- If **var_type** is 30:

01 02	30	.. ..	00	.. ..	.. ..	..
command_code	**var_ type**	**start_ address**	**bit_ num**	**total_bits**	**bit**	**..**

The parameters can have the following values:

Parameter	Values
command_code	01 02
var_type	• 82 (Table memory in 16-bit integer format) • C2 (Table memory in 32-bit IEEE floating-point format) • B0 (VR memory in 16-bit integer format) • F0 (VR memory in 32-bit IEEE floating-point format) • 30 (VR memory in bit format)
start_address	0 ≤ **start_address** ≤ memory size – 1 ≤ FFFF
total_words	1 ≤ **total_words** ≤ memory size - **start_address** + 1
total_dwords	1 ≤ **total_dwords** ≤ memory size - **start_address** + 1
total_bits	1
bit	00 or 01

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
var_type invalid	1101	No area type
start_address invalid	1103	Address range designation error
bit_number invalid	1103	Address range designation error
Number of elements invalid (totals)	1104	Address out of range

3-4-3 Parameter Area Read (0201)

The FINS Parameter Area Read command reads the memory-mapping configuration that is written with the FINS Parameter Area Write command (see section 3-4-4).
The Parameter Area Read command has this format:

02 01	.. ..	00 00	00 08
command_code	**area_code**	**start_address**	**byte_count**

The parameters can have the following values:

Parameter	Values (hex)
command_code	02 01
area_code	• 0100..0107 for PLC output area (8 areas available) • 8100..8107 for PLC input area (8 areas available)
start_address	0000
byte_count	0008

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
area_code invalid	1101	No area type
start_address invalid	1103	Address range designation error
byte_count invalid	1104	Address out of range
area_code not configured	2003	The registered table does not exist

If the response code is 0000, the CJ1W-MCH72 responds with the data configured previously:

02 01	00 00	.. ..	00 00	00 08	..	.. ..	..	.. ..	.. ..
command_ code	**response_ code**	**area_ code**	**start_ address**	**byte_ count**	**plc_ area**	**plc_ start**	**tj_ area**	**tj_ start**	**total_ items**

Refer to section 3-4-4 for more information on the fields **plc_area**, **plc_start**, **tj_area**, **tj_start** and **total_items**.

3-4-4 Parameter Area Write (0202)

The FINS Parameter Area Write command sets the memory-mapping configuration. It has this format:

02 02	00 00	.. ..	00 00	00 08	..	.. ..	..	.. ..	.. ..
command_ code	**response_ code**	**area_ code**	**start_ address**	**byte_ count**	**plc_ area**	**plc_ start**	**tj_ area**	**tj_ start**	**total_ items**

The parameters can have the following values:

Parameter	Values (hex)
command_code	02 02
area_code	• 0100..0107 for PLC output area (8 areas available) • 8100..8107 for PLC input area (8 areas available)
start_address	0000
byte_count	0008
plc_area	• 01 (CIO) • 03 (DM) • 04 (WR) • 05 (HR) • 08..14 (EM bank 0..C)
plc_start	Start address in PLC memory (Validity depends on **plc_area**)
tj_area	• 00 (VR 16-bit signed integer) • 01 (VR 32-bit floating point) • 02 (IN or OP array, depending on direction) • 03 (AIN or AOUT array, depending on direction) • 04 (Axis Status array, only valid if direction is PLC input)
tj_start	Start address in CJ1W-MCH72 memory (Validity depends on **tj_area**)
total_items	Total items (words and dwords) to transfer (Validity depends on **plc_area** and **tj_area**)

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
area_code invalid	1101	No area type
start_address invalid	1103	Address range designation error
byte_count invalid	1104	Address out of range
plc_area, plc_start, **tj_area** or **tj_start** invalid	110C	Other parameter error
total_items greater than 2000 words	2103	Data has not been registered

If the response code is 0000, the cyclic area number specified in **area_code** is configured to exchange data between the PLC CPU and the CJ1W-MCH72.

3-4-5 Run (0401)

The FINS Run command starts or stops a BASIC program.

The FINS Run command has these formats:

- To start a BASIC program:

04	01	00	..	01	..	..
command_code		**process**		**mode**	**program_name**	

- To stop a BASIC program:

04	01	00	..	00
command_code		**process**		**mode**

The parameters can have the following values:

Parameter	Values
mode	• 00 (Stop) • 01 (Start)
process	01..0E (Process number)
program_name	A string that represents the program name (The string does not have zero termination)

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
process invalid	1106	Process number does not exist or invalid mode
mode invalid		
program_name invalid	2402	Program name does not exist
Start BASIC program that is already running	2201	Wrong mode (executing)
Stop BASIC program that is not running	2202	Wrong mode (stopped)

If the response code is 0000, the program is started or stopped.

3-4-6 Stop (0402)

The FINS Stop command stops a BASIC program. It has this format:

04	02	00	..
command_code		**process**	

The parameters can have the following values:

Parameter	Values
process	01..0E (Process number)

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK
process invalid	1106	Process number does not exist
Stop BASIC program that is not running	2202	Wrong mode (stopped)

Note The FINS Stop command (0402) is equal to the FINS Run command (0401) where **mode** is equal to 00.

3-4-7 Error Data Read (2110)

The FINS Error Data Read command reads the error data (error line and error code) of a process. It has this format:

21 10	00 ..
command_code	**process**

The parameters can have the following values:

Parameter	Values
process	01..0E (Process number)

The CJ1W-MCH72 responds with these codes:

Condition	Response code (hex)	Description
All elements valid	0000	OK

If the response code is 0000, the CJ1W-MCH72 responds with the error data:

21 01	00 00	.. ..	.. ..
command_code	**response_code**	**error_line**	**error_code**

error_line is equal to the return value of the BASIC command **ERROR_LINE PROC(process)**.
error_code is equal to the return value of the BASIC command **RUN_ERROR PROC(process)**.

SECTION 4
BASIC commands

4-1 Categories

This section lists all BASIC commands divided by categories. The categories are:

- Axis commands.
- Axis parameters.
- Communication commands and parameters.
- Constants.
- I/O commands, functions and parameters.
- Mathematical functions and operations.
- Program commands.
- Program control commands.
- Slot parameters and modifiers.
- System commands and functions.
- System parameters.
- Task commands and parameters.

The lists are quick reference guides only. A complete description of the commands is given in alphabetical order in the next section.

4-1-1 Axis commands

Name	Description
ACC	Changes the **ACCEL** and **DECEL** at the same time.
ADD_DAC	Sum to the **S_REF** value of one axis to the analogue output of the base axis.
ADDAX	Sets a link to a superimposed axis. All demand position movements for the superimposed axis will be added to any moves that are currently being executed.
B_SPLINE	Expands the profile stored in **TABLE** memory using the B-Spline mathematical function.
BACKLASH	Allows the backlash compensation to be loaded.
BASE	Used to set the base axis to which the commands and parameters are applied.
CAM	Moves an axis according to values of a movement profile stored in the TABLE variable array.
CAMBOX	Moves an axis according to values of a movement profile stored in the TABLE variable array. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox.
CANCEL	Cancels the move on an axis.
CONNECT	Connects the demand position of an axis to the measured movements of the axis specified for **driving_axis** to produce an electronic gearbox.
DATUM	Performs one of 7 origin search sequences to position an axis to an absolute position or reset a motion error.
DEFPOS	Defines the current position as a new absolute position.

Name	Description
DISABLE_GROUP	Groups axes together for error disabling.
DRIVE_ALARM	Monitors the current alarm.
DRIVE_CLEAR	Clears the alarm status of the Servo Driver.
DRIVE_READ	Reads the specified parameter of the Servo Driver.
DRIVE_RESET	Resets the Servo Driver.
DRIVE_WRITE	Writes a specific value to the specified parameter of the Servo Driver.
ENCODER_READ	Reads a parameter of the EnDat absolute encoder.
ENCODER_WRITE	Writes to a parameter of the EnDat absolute encoder.
FORWARD	Moves an axis continuously forward at the speed set in the **SPEED** parameter.
HW_PSWITCH	Sets on and off the hardware switch on output 0 of the Encoder Interface when predefined positions are reached.
MECHATROLINK	Initializes MECHATROLINK-II bus and performs various operations on MECHATROLINK-II stations connected to the bus.
MHELICAL	Interpolates 3 orthogonal axes in a helical move.
MOVE	Moves one or more axes at the demand speed, acceleration and deceleration to the position specified as increment from the current position.
MOVEABS	Moves one or more axes at the demand speed, acceleration and deceleration to the position specified as absolute position.
MOVECIRC	Interpolates 2 orthogonal axes in a circular arc.
MOVELINK	Creates a linear move on the base axis linked via a software gearbox to the measured position of a link axis.
MOVEMODIFY	Changes the absolute end position of the current single-axis linear move (**MOVE** or **MOVEABS**).
RAPIDSTOP	Cancels the current move on all axes.
REGIST	Captures an axis position when a registration input or the Z mark on the encoder is detected.
REVERSE	Moves an axis continuously in reverse at the speed set in the **SPEED** parameter.
STEP_RATIO	Sets the ratio for the axis stepper output.

4-1-2 Axis parameters

Name	Description
ACCEL	Contains the axis acceleration rate.
ADDAX_AXIS	Contains the number of the axis to which the base axis is currently linked to by **ADDAX**.
ATYPE	Contains the axis type.
AXIS_ENABLE	Enables and disables particular axis independently of other axis.

Name	Description
AXISSTATUS	Contains the axis status.
BACKLASH_DIST	Defines the amount of backlash compensation.
CLOSE_WIN	Defines the end of the window in which a registration mark is expected.
CLUTCH_RATE	Defines the change in connection ratio when using the **CONNECT** command.
CREEP	Contains the creep speed.
D_GAIN	Contains the derivative control gain.
DATUM_IN	Contains the input number to be used as the origin input.
DECEL	Contains the axis deceleration rate.
DEMAND_EDGES	Contains the current value of the **DPOS** axis parameter in encoder edges.
DPOS	Contains the demand position generated by the move commands.
DRIVE_CONTROL	Selects data to be monitored using **DRIVE_MONITOR** for axes connected via the MECHATROLINK-II bus. For axes connected via the Encoder Interface, **DRIVE_CONTROL** sets outputs of the Encoder Interface.
DRIVE_INPUTS	Holds I/O data of the driver connected to MECHATROLINK-II bus. Data is updated every servo cycle.
DRIVE_MONITOR	Monitors data of the Servo Driver connected to MECHATROLINK-II bus. Data are updated every servo cycle.
DRIVE_STATUS	Contains the current status of the Servo Driver.
ENCODER	Contains a raw copy of the encoder hardware register.
ENCODER_BITS	Sets the number of bits for the absolute encoder connected to the Encoder Interface.
ENCODER_CONTROL	Controls operating mode of the EnDat absolute encoder.
ENCODER_RATIO	Sets scaling value for incoming encoder counts.
ENCODER_TURNS	Returns the multi-turn count of the absolute encoder.
ENDMOVE	Holds the position of the end of the current move.
ERRORMASK	Contains the mask value that determines if **MOTION_ERROR** occurs depending on the axis status.
FAST_JOG	Contains the input number to be used as the fast jog input.
FASTDEC	Defines ramp to zero deceleration ratio when an axis limit switch or position is reached.
FE	Contains the Following Error.
FE_LATCH	Contains the FE value which caused the axis to put controller in **MOTION_ERROR** state.
FE_LIMIT	Contains the maximum allowable Following Error.
FE_LIMIT_MODE	Defines how FE influences **MOTION_ERROR** state.
FE_RANGE	Contains the Following Error warning range limit.
FHOLD_IN	Contains the input number to be used as the feedhold input.

Name	Description
FHSPEED	Contains the feedhold speed.
FS_LIMIT	Contains the absolute position of the forward software limit.
FWD_IN	Contains the input number to be used as a forward limit input.
FWD_JOG	Contains the input number to be used as a jog forward input.
I_GAIN	Contains the integral control gain.
INVERT_STEP	Switches a hardware Inverter into the stepper output circuit.
JOGSPEED	Sets the jog speed.
LINKAX	Contains the axis number of the link axis during any linked move.
MARK	Detects the primary registration event on a registration input.
MARKB	Detects the secondary registration event on a registration input.
MERGE	Is a software switch that can be used to enable or disable the merging of consecutive moves.
MPOS	Is the position of the axis as measured by the encoder.
MSPEED	Represents the change in the measured position in the last servo period.
MTYPE	Contains the type of move currently being executed.
NTYPE	Contains the type of the move in the Next Move buffer.
OFFPOS	Contains an offset that will be applied to the demand position without affecting the move in any other way.
OPEN_WIN	Defines the beginning of the window in which a registration mark is expected.
OUTLIMIT	Contains the limit that restricts the speed reference output from the CJ1W-MCH72.
OV_GAIN	Contains the output velocity control gain.
P_GAIN	Contains the proportional control gain.
REG_POS	Contains the position at which a registration event occurred.
REG_POSB	Contains the position at which the secondary registration event occurred.
REMAIN	Is the distance remaining to the end of the current move.
REMOTE_ERROR	Returns number of errors on MECHATROLINK-II connection of the Servo Driver.
REP_DIST	Contains or sets the repeat distance.
REP_OPTION	Controls the application of the **REP_DIST** axis parameter.
REV_IN	Contains the input number to be used as a reverse limit input.

Name	Description
REV_JOG	Contains the input number to be used as a jog reverse input.
RS_LIMIT	Contains the absolute position of the reverse software limit.
S_REF	Contains the speed reference value which is applied when the axis is in open loop.
S_REF_OUT	Contains the speed reference value being applied to the Servo Driver for both open as closed loop.
SERVO	Determines whether the axis runs under servo control or open loop.
SPEED	Contains the demand speed in units/s.
SPEED_SIGN	Configures the voltage range of the analogue speed reference output of the Encoder Interface.
SRAMP	Contains the S-curve factor.
T_REF	Contains the torque reference value which is applied to the servo motor.
TRANS_DPOS	Contains axis demand position at output of frame transformation.
UNITS	Contains the unit conversion factor.
VERIFY	Selects different modes of operation on a stepper output axis.
VFF_GAIN	Contains the speed feed forward control gain.
VP_SPEED	Contains the speed profile speed.

4-1-3 Communication commands and parameters

Name	Description
FINS_COMMS	Sends FINS Read Memory and Write Memory to a designated FINS server unit.

4-1-4 Constants

Name	Description
FALSE	Equal to the numerical value 0.
OFF	Equal to the numerical value 0.
ON	Equal to the numerical value 1.
PI	Equal to the numerical value 3.1416.
TRUE	Equal to the numerical value -1.

4-1-5 I/O commands, functions and parameters

Name	Description
AIN	Holds the value of the analogue channel.

Name	Description
AOUT	Holds the value of the analogue channel.
GET	Waits for the arrival of a single character and assigns the ASCII code of the character to variable.
IN	Returns the value of digital inputs.
OP	Sets one or more outputs or returns the state of the first 24 outputs.
PRINT	Outputs a series of characters to a serial port.
PSWITCH	Turns on an output when a predefined position is reached, and turns off the output when a second position is reached.
READ_OP	Returns the value of the digital outputs.

4-1-6 Mathematical functions and operands

Name	Description
+ (ADDITION)	Adds two expressions.
- (SUBTRACTION)	Subtracts two expressions.
*** (MULTIPLICATION)**	Multiplies two expressions.
/ (DIVISION)	Divides two expressions.
^ (POWER)	Takes the power of one expression to the other expression.
= (IS EQUAL TO)	Checks two expressions to see if they are equal.
= (ASSIGNMENT)	Assigns an expression to a variable.
<> (IS NOT EQUAL TO)	Checks two expressions to see if they are different.
> (IS GREATER THAN)	Checks two expressions to see if the expression on the left is greater than the expression on the right.
>= (IS GREATER THAN OR EQUAL TO)	Checks two expressions to see if the expression on the left is greater than or equal to the expression on the right.
< (IS LESS THAN)	Checks two expressions to see if the expression on the left is less than the expression on the right.
<= (IS LESS THAN OR EQUAL TO)	Checks two expressions to see if the expression on the left is less than or equal to the expression on the right.
ABS	Returns the absolute value of an expression.
ACOS	Returns the arc-cosine of an expression.
AND	Performs an AND operation on corresponding bits of the integer parts of two expressions.
ASIN	Returns the arc-sine of an expression.
ATAN	Returns the arc-tangent of an expression.
ATAN2	Returns the arc-tangent of the non-zero complex number made by two expressions.
COS	Returns the cosine of an expression.
EXP	Returns the exponential value of an expression.

Name	Description
FRAC	Returns the fractional part of an expression.
IEEE_IN	Returns floating point number in IEEE format, represented by 4 bytes.
IEEE_OUT	Returns single byte extracted from the floating point number in IEEE format.
INT	Returns the integer part of an expression.
LN	Returns the natural logarithm of an expression.
MOD	Returns the modulus of two expressions.
NOT	Performs a NOT operation on corresponding bits of the integer part of the expression.
OR	Performs an OR operation between corresponding bits of the integer parts of two expressions.
SGN	Returns the sign of an expression.
SIN	Returns the sine of an expression.
SQR	Returns the square root of an expression.
TAN	Returns the tangent of an expression.
XOR	Performs an XOR function between corresponding bits of the integer parts of two expressions.

4-1-7 Program commands

Name	Description
' (COMMENT FIELD)	Enables a line not to be executed.
: (STATEMENT SEPARATOR)	Enables more statements on one line.
AUTORUN	Starts all the programs that have been set to run at start-up.
COMPILE	Compiles the current program.
COPY	Copies an existing program in the motion controller to a new program.
DEL	Deletes a program from the motion controller.
DIR	Displays a list of the programs in the motion controller, their size and their RUNTYPE on the standard output.
EDIT	Allows a program to be modified using a VT100 Terminal.
EPROM	Stores a program in the flash memory.
LIST	Prints the program on the standard output.
NEW	Deletes all lines of the program in the motion controller.
PROCESS	Returns the running status and task number for each current task.
RENAME	Changes the name of a program in the motion controller.

Name	Description
RUN	Executes a program.
RUNTYPE	Determines if a program is run at start-up, and which task it is to run on.
SELECT	Specifies the current program.
STEPLINE	Executes a single line in a program.
STOP	Halts program execution.
TROFF	Suspends a trace at the current line and resumes normal program execution.
TRON	Creates a breakpoint in a program.

4-1-8 Program control commands

Name	Description
FOR..TO..STEP..NEXT	Loop allows a program segment to be repeated with increasing/decreasing variable.
GOSUB..RETURN	Jumps to a subroutine at the line just after label. The program execution returns to the next instruction after a “RETURN” on page 214 is given.
GOTO	Jumps to the line containing the label.
IF..THEN..ELSE..ENDIF	Controls the flow of the program base on the results of the condition.
ON.. GOSUB or **ON.. GOTO**	Enables a conditional jump to one of several labels.
REPEAT..UNTIL	Loop allows the program segment to be repeated until the condition becomes “TRUE” on page 235.
WHILE..WEND	Loop allows the program segment to be repeated until the condition becomes **FALSE**.

4-1-9 Slot parameters and modifiers

Name	Description
ALL	Is a modifier that specifies that all items in the controller are concerned.
FPGA_VERSION	Returns the FPGA version.

4-1-10 System commands and functions

Name	Description
$ (HEXADECIMAL INPUT)	Assigns a hexadecimal number to a variable.
AXIS	Sets the axis for a command, axis parameter read, or assignment to a particular axis.
BASICERROR	Is used to run a specific routine when an error occurs in a BASIC command.

Name	Description
CLEAR	Clears all global variables and the local variables on the current task.
CLEAR_BIT	Clears the specified bit of the specified VR variable.
CLEAR_PARAMS	Clears all parameter sand variables stored in flash EPROM to their default values.
CONSTANT	Declares a constant for use in BASIC program.
DATE$	Prints the current date as a string.
EX	Resets the controller.
FLAG	Sets and reads a bank of 32 bits.
FLAGS	Read and sets FLAGS as a block.
FREE	Returns the amount of available memory.
GLOBAL	Declares a reference to one of VR variables.
HALT	Stops execution of all programs currently running.
INITIALISE	Sets all axes and parameters to their default values.
INVERT_IN	Inverts input channels 0 - 31 in the software.
INVERTER_COMMAND	Reads I/O and clears alarm of the Inverter.
INVERTER_READ	Reads parameter, alarm, speed and torque reference of the Inverter.
INVERTER_WRITE	Writes to parameter, speed and torque reference of the Inverter.
LIST_GLOBAL	Shows all GLOBAL and CONSTANT variables.
LOCK	Prevents the programs from being viewed or modified.
PLC_EXCHANGE	Reads or sets the mapping of PLC memory to CJ1W-MCH72 memory.
READ_BIT	Returns the value of the specified bit in the specified VR variable.
RESET	Resets all local variables on a task.
SCOPE	Programs the system to automatically store up to 4 parameters every sample period to the TABLE variable array.
SET_BIT	Sets the specified bit in the specified VR variable to one.
TABLE	Writes and reads data to and from the TABLE variable array.
TABLEVALUES	Returns list of values from the TABLE memory.
TIME$	Prints the current time as a string.
TRIGGER	Starts a previously set **SCOPE** command.
VR	Writes and reads data to and from the global (VR) variables.
VRSTRING	Combines VR memory values so they can be printed as a string.

Name	Description
WA	Holds program execution for the number of milliseconds specified.
WAIT IDLE	Suspends program execution until the base axis has finished executing its current move and any buffered move.
WAIT LOADED	Suspends program execution until the base axis has no moves buffered ahead other than the currently executing move.
WAIT UNTIL	Repeatedly evaluates the condition until it is **TRUE**.

4-1-11 System parameters

Name	Description
BATTERY_LOW	Returns the current status of the battery condition.
CHECKSUM	Contains the checksum for the programs in RAM.
CONTROL	Contains the type of controller in the system.
D_ZONE_MAX	Controls the S_REF output in conjunction with the Following Error value.
D_ZONE_MIN	Controls the S_REF output in conjunction with the Following Error value.
DATE	Sets or returns the current date held by the real time clock.
ERROR_AXIS	Contains the number of the axis which caused the motion error.
FRAME	Specifies operating frame for frame transformations.
LAST_AXIS	Contains the number of the last axis processed by the system.
MOTION_ERROR	Contains an error flag for axis motion errors.
NEG_OFFSET	Applies a negative offset to the S_REF signal from the servo loop.
PLC_STATUS	Contains the PLC status.
POWER_UP	Determines whether programs should be read from flash EPROM on power up or reset.
POS_OFFSET	Applies a positive offset to the S_REF signal from the servo loop.
SCOPE_POS	Contains the current TABLE position at which the SCOPE command is currently storing its first parameter.
SERVO_PERIOD	Sets the servo cycle period of the CJ1W-MCH72.
SYSTEM_ERROR	Contains the system errors since the last initialization.
TIME	Returns the current time held by the real time clock.
TSIZE	Returns the size of the currently defined Table.
VERSION	Returns the version number of the controller firmware.
WDOG	The software switch that enables Servo Drivers.

4-1-12 Task commands and parameters

Name	Description
ERROR_LINE	Contains the number of the line which caused the last BASIC program error.
PMOVE	Contains the status of the task buffers.
PROC	Lets a process parameter from a particular process to be accessed.
PROC_STATUS	Returns the status of the process specified.
PROCNUMBER	Contains the number of the task in which the currently selected program is running.
RUN_ERROR	Contains the number of the last BASIC error that occurred on the specified task.
TICKS	Contains the current count of the task clock pulses.

4-2 All BASIC commands

4-2-1 + (Addition)

Type Mathematical function

Syntax **expression1 + expression2**

Description The operator **+** adds two expressions.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 4 + 3**
Assigns the value 7 to the variable **result**.

See also N/A

4-2-2 - (Subtraction)

Type Mathematical function

Syntax **expression1 - expression2**

Description The operator **-** subtracts **expression2** from **expression1**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 10 - 2**
Assigns the value 8 to the variable **result**.

See also N/A

4-2-3 * (Multiplication)

Type Mathematical function

Syntax **expression1 * expression2**

Description The operator ***** multiplies two expressions.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 3 * 7**
Assigns the value 21 to the variable **result**.

See also N/A

4-2-4 / (Division)

Type Mathematical function

Syntax **expression1 / expression2**

Description The operator **/** divides **expression1** by **expression2**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 11 / 4**
Assigns the value 2.75 to the variable **result**.

See also N/A

4-2-5 ^ (Power)

Type Mathematical function

Syntax **expression1 ^ expression2**

Description The power operator **^** raises **expression1** to the power of **expression2**.
This operation uses floating point algorithms and may give small deviations for integer calculations.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 2^5**
Assigns the value 32 to the variable **result**.

See also N/A

4-2-6 = (Is equal to)

Type Mathematical function

Syntax **expression1 = expression2**

Description The operator **=** returns **TRUE** if **expression1** is equal to **expression2**, otherwise it returns **FALSE**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **IF a = 10 THEN GOTO label1**
If variable **a** contains a value equal to 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-7 = (Assignment)

Type Mathematical function

Syntax **variable = expression**

Description The operator **=** assigns the value of the expression to the variable.

Arguments
- **variable**
  A variable name.
- **expression**
  Any valid BASIC expression.

Example **var = 18**
Assigns the value 18 to the variable **var**.

See also N/A

4-2-8 <> (Is not equal to)

Type Mathematical function

Syntax **expression1 <> expression2**

Description The operator **<>** returns **TRUE** if **expression1** is not equal to **expression2**, otherwise it returns **FALSE**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **IF a <> 10 THEN GOTO label1**
If variable **a** contains a value not equal to 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-9 > (Is greater than)

Type Mathematical function

Syntax **expression1 > expression2**

Description The operator **>** returns **TRUE** if **expression1** is greater than **expression2**, otherwise it returns **FALSE**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **IF a > 10 THEN GOTO label1**
If variable **a** contains a value greater than 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-10 >= (Is greater than or equal to)

Type Mathematical function

Syntax **expression1 >= expression2**

Description The operator **>**= returns **TRUE** if **expression1** is greater than or equal to **expression2**, otherwise it returns **FALSE**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **IF a >=10 THEN GOTO label1**
If variable **a** contains a value greater than or equal to 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-11 < (Is less than)

Type Mathematical function

Syntax **expression1 < expression2**

Description The operator **<** returns **TRUE** if **expression1** is less than **expression2**, otherwise it returns **FALSE**.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **IF a < 10 THEN GOTO label1**
If variable **a** contains a value less than 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-12 <= (Is less than or equal to)

Type	Mathematical function
Syntax	**expression1 <= expression2**
Description	The operator **<=** returns **TRUE** if **expression1** is less than or equal to **expression2**, otherwise it returns **FALSE**.
Arguments	• **expression1** Any valid BASIC expression. • **expression2** Any valid BASIC expression.
Example	**IF a <= 10 THEN GOTO label1** If variable **a** contains a value less than or equal to 10, program execution continues at label **label1**. Otherwise, program execution continues with the next statement.
See also	N/A

4-2-13 $ (Hexadecimal input)

Type	System command
Syntax	**$hex_num**
Description	The **$** command makes the number that follows a hexadecimal number.
Arguments	• **hex_num** A hexadecimal number (consisting of the characters 0 - 9 and A - F). **hex_num** ranges from 0 to FFFFFF.
Example	**>>TABLE(0, $F, $ABCD)** **>>print TABLE(0), TABLE(1)** **15.0000 43981.0000**
See also	**HEX (PRINT)**

4-2-14 ' (Comment field)

Type	Program command
Syntax	**'**
Description	**'** marks all that follows it on a line as comment and not program code. Comment is not executed when the program is run. You can use **'** at the beginning of a line or after a valid statement.
Arguments	N/A
Example	**' This line is not printed** **PRINT "Start"**
See also	N/A

4-2-15 : (Statement separator)

Type	Program command
Syntax	**:**

Description The statement separator **:** separates multiple BASIC statements on one line. You can use it on the command line and in programs.

Arguments N/A

Example **PRINT "THIS LINE": GET low : PRINT "DOES THREE THINGS"**

See also N/A

4-2-16

Type Special character

Syntax #

Description The # symbol is used to specify a communications channel to be used for serial input/output commands.
Note: Communications Channels greater than 3 will only be used when running the Trajexia Studio software.

Arguments N/A

Example **PRINT #1, "RS232"**
PRINT #2, "RS485"

See also N/A

4-2-17 ABS

Type Mathematical function

Syntax **ABS(expression)**

Description The **ABS** function returns the absolute value of an expression.

Arguments
- **expression**
  Any valid BASIC expression.

Example **IF ABS(A) > 100 THEN PRINT "A is outside range -100 ... 100"**

See also N/A

4-2-18 ACC

Type Axis command

Syntax **ACC(rate)**

Description Sets the acceleration and deceleration at the same time.
This command gives a quick method to set both **ACCEL** and **DECEL**. Acceleration and deceleration rates are recommended to be set with the **ACCEL** and **DECEL** axis parameters.

Arguments
- **rate**
  The acceleration/deceleration rate in units/s^2. You can define the units with the **UNITS** axis parameter.

Example **ACC(100)**
Sets **ACCEL** and **DECEL** to 100 units/s^2.

See also **ACCEL**, **DECEL**, **UNITS**

4-2-19 ACCEL

Type Axis parameter

Syntax **ACCEL = expression**

Description The **ACCEL** axis parameter contains the axis acceleration rate. The rate is set in units/s^2. The parameter can have any positive value including zero.

Arguments N/A

Example
```
BASE(0)
ACCEL = 100 ' Set acceleration rate
PRINT "Acceleration rate: "; ACCEL; " mm/s/s"
ACCEL AXIS(2) = 100 ' Sets acceleration rate for axis (2)
```

See also **ACCEL**, **DECEL**, **UNITS**

4-2-20 ACOS

Type Mathematical function

Syntax **ACOS(expression)**

Description The **ACOS** function returns the arc-cosine of the expression. The expression value must be between -1 and 1. The result in radians is between 0 and PI. Input values outside the range will return 0.

Arguments
- **expression**
  Any valid BASIC expression.

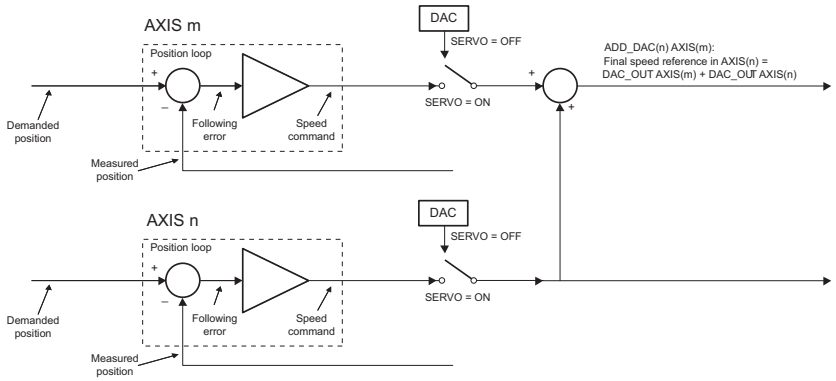
Example
```
>> PRINT ACOS(-1)
3.1416
```

See also N/A

4-2-21 ADD_DAC

Type Axis command

Syntax **ADD_DAC(axis)**



Description The **ADD_DAC** command adds the **S_REF_OUT** value of **axis** to the **S_REF_OUT** value of the base axis. Use **ADD_DAC(-1)** to cancel the sum.
ADD_DAC works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note:
1 Be aware that the control loop gains for both axes need to be determined with care. As different encoders with different resolutions are used, the gains are not identical.
2 Set the **OUTLIMIT** parameter to the same value for both linked axes.
3 This command has no meaning for a MECHATROLINK-II axis in position mode (**ATYPE=40**), because the value of **S_REF_OUT** is ignored.

Arguments
- **axis**
  The axis from which to sum the speed reference output to the base axis. Set the argument to -1 to cancel the link and return to normal operation.

Example No example.

See also **AXIS**, **S_REF_OUT**, **OUTLIMIT**

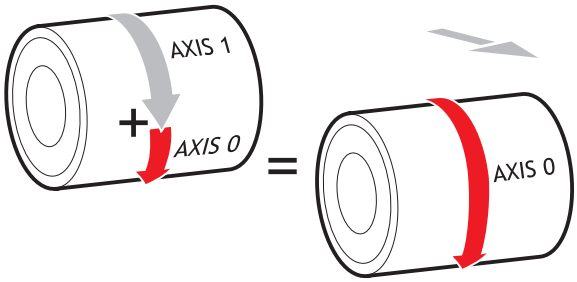
4-2-22 ADDAX

Type Axis command

Syntax **ADDAX(axis)**

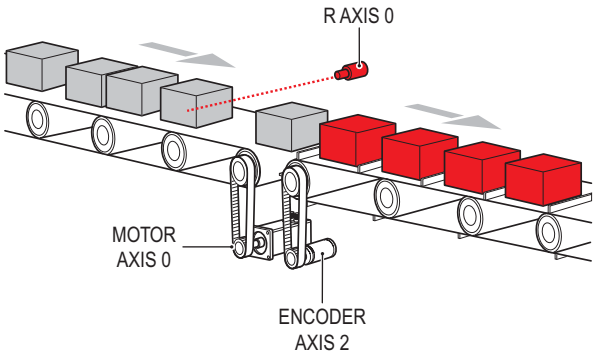
Description The **ADDAX** command is used to superimpose two or more movements to build up a more complex movement profile.
The **ADDAX** command takes the demand position changes from the superimposed axis as specified by the axis argument and adds them to any movement running on the axis to which the command is issued. The axis specified by the parameter can be any axis and does not have to physically exist in the system.
The **ADDAX** command therefore allows an axis to perform the moves specified on two axes added together. When the axis parameter is set to OFF on an axis with an encoder interface the measured position **MPOS** is copied into the demanded position **DPOS**. This allows **ADDAX** to be used to sum encoder inputs.
After the **ADDAX** command has been issued the link between the two axes remains until broken. Use **ADDAX(-1)** to cancel the axis link. **ADDAX** allows an axis to perform the moves specified for 2 axes added together. Combinations of more than two axes can be made by applying **ADDAX** to the superimposed axis as well.
ADDAX works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note: The **ADDAX** command sums the movements in encoder edge units.

Arguments
- **axis**
  The axis to be set as a superimposed axis. Set the argument to -1 to cancel the link and return to normal operation.



Example

```
UNITS AXIS(0)=1000
UNITS AXIS(1)=20
' Superimpose axis 1 on axis 0
ADDAX(1) AXIS(0)
MOVE(1) AXIS(0)
MOVE(2) AXIS(1)
'Axis 0 will move 1*1000+2*20=1040 edges
```

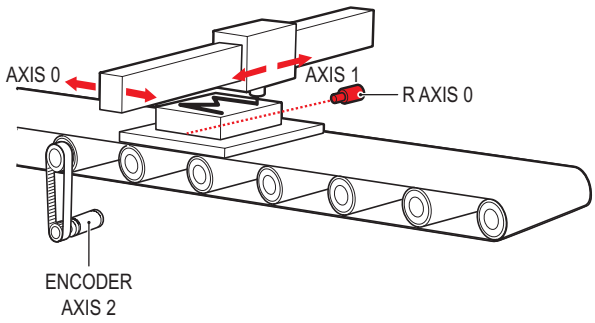


Example

Pieces are placed randomly onto a belt that moves continuously. Further along the line they are transferred to a second flighted belt. A detection system indicates if a piece is in front of or behind its nominal position, and how far.

```
expected=2000 ' sets expected position
BASE(0)
ADDAX(1)
CONNECT(1,2) ' continuous geared connection to flighted belt
REPEAT
  GOSUB getoffset ' get offset to apply
  MOVE(offset) AXIS(1) ' make correcting move on virtual axis
UNTIL IN(2)=OFF ' repeat until stop signal on input 2
RAPIDSTOP
ADDAX(-1) ' clear ADDAX connection
STOP
getoffset: ' sub routine to register the position of the
  ' piece and calculate the offset
BASE(0)
REGIST(3)
WAIT UNTIL MARK
seenat=REG_POS
offset=expected-seenat
RETURN
```

Axis 0 in this example is connected to the encoder of the second conveyor. A superimposed **MOVE** on axis 1 is used to apply offsets.



Example An X-Y marking machine must mark boxes as they move along a conveyor. Using CONNECT enables the X marking axis to follow the conveyor. A virtual axis is used to program the marking absolute positions; this is then superimposed onto the X axis using ADDAX.

```
ATYPE AXIS(3)=0 'set axis 3 as virtual axis
SERVO AXIS(3)=ON
DEFPOS(0) AXIS(3)
ADDAX (3)AXIS(0) 'connect axis 3 requirement to axis 0
WHILE IN(2)=ON
 REGIST(3) 'registration input detects a box on the conveyor
 WAIT UNTIL MARK OR IN(2)=OFF
 IF MARK THEN
  CONNECT(1,2) AXIS(0)'connect axis 0 to the moving belt
  BASE(3,1) 'set the drawing motion to axis 3 and 1
  'Draw the M
  MOVEABS(1200,0)'move A > B
  MOVEABS(600,1500)'move B > C
  MOVEABS(1200,3000)' move C > D
  MOVEABS(0,0)'move D > E
  WAIT IDLE
  BASE(0)
  CANCEL 'stop axis 0 from folowing the belt
  WAIT IDLE
  MOVEABS(0) 'move axis 0 to home position
 ENDIF
WEND
CANCEL
```

See also **ADDAX_AXIS**, **AXIS**, **OUTLIMIT**

⚠ **WARNING** Beware that giving several **ADDAX** commands in a system can create a dangerous loop when for instance one axis is linked to another and vice versa. This may cause instability in the system.

4-2-23 ADDAX_AXIS

Type Axis parameter (read-only)

Syntax **ADDAX_AXIS**

Description The **ADDAX_AXIS** axis parameter returns the number of the axis to which the base axis is currently linked to by **ADDAX**. If the base axis in not linked to any other axis, the **ADDAX_AXIS** parameter returns -1.

Arguments N/A

Example

```
>> BASE(0)
>> ADDAX(2)
>> PRINT ADDAX_AXIS
2.0000
>> ADDAX(-1)
>> PRINT ADDAX_AXIS
-1.0000
```

See also **ADDAX**, **AXIS**

4-2-24 AIN

Type I/O command

Syntax **AIN(analogue_chan)**

Description The AIN reads a value from an analogue input. Analogue input channels are provided by the analogue input modules connected to the PLC.

Arguments **analogue_chan**.
Analogue input channel number 0..31

Example
```
MOVE(-5000)
REPEAT
  a=AIN(1)
  IF a<0 THEN a=0
  SPEED=a*0.25
UNTIL MTYPE=0
```
The speed of a production line is governed by the rate at which material is fed onto it. The material feed is via a lazy loop arrangement which is fitted with an ultra-sonic height sensing device. The output of the ultra-sonic sensor is in the range 0V to 4V where the output is at 4V when the loop is at its longest.
Note: The analogue input value is checked to ensure it is above zero even though it always should be positive. This is to allow for any noise on the incoming signal which could make the value negative and cause an error because a negative speed is not valid for any move type except **FORWARD** or **REVERSE**.

See also N/A

4-2-25 ALL

Type Slot modifier

Syntax **ALL**

Description The ALL modifier is used with the commands **DEL** and **NEW**. It indicates that these commands are applied to all items in the directory structure of the controller.

Arguments N/A

Example
```
DEL ALL
```
This deletes all programs and the TABLE memory of the controller.

Example
```
HALT
NEW ALL
STORE
```
This creates the memory of the controller.

See also **DEL**, **NEW**.

4-2-26 AND

Type Mathematical operation

Syntax **expression1 AND expression2**

Description The **AND** operator performs the logical **AND** function on the corresponding bits of the integer parts of two valid BASIC expressions.
The logical **AND** function between two bits is defined as follows:
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **VR(0) = 10 AND (2.1*9)**
The parentheses are evaluated first, but only the integer part of the result, 18, is used for the **AND** operation. Therefore, this expression is equivalent to the following:
VR(0) = 10 AND 18
The **AND** is a bit operator and so the binary action is as follows:
01010 AND 10010 = 00010
Therefore, **VR(0)** will contain the value 2.

Example **IF MPOS AXIS(0) > 0 AND MPOS AXIS(1) > 0 THEN GOTO cycle1**
If measured positions **MPOS** of both axis 1 and axis 2 are greater than zero, program execution continues at label **cycle1**. Otherwise, program execution continues with the next statement.

See also N/A

4-2-27 AOUT

Type I/O command

Syntax **AOUT(analogue_chan)**

Description The **AOUT** command sets the output value of the analogue output channels that are provided by the analogue output modules connected to the PLC. The range of the value set is [-32000, 32000] for full output range. The output range depends on the analogue unit used and can be one of the following: [-10V, 10V], [0V, 10V] or [0V, 5V] for voltage and [0mA, 20mA] or [4mA, 20mA] for current output.

Arguments
- **analogue_chan**.
  Analogue output channel number 0.31

Example No example.

See also N/A

4-2-28 ASIN

Type Mathematical function

Syntax **ASIN(expression)**

Description The **ASIN** function returns the arc-sine of the argument. The argument must have a value between -1 and 1. The result in radians is between -PI/2 and PI/2. Input values outside this range return 0.

Arguments
- **expression**
  Any valid BASIC expression.

Example **>> PRINT ASIN(-1)**
-1.5708

See also N/A

4-2-29 ATAN

Type Mathematical function

Syntax **ATAN(expression)**

Description The **ATAN** function returns the arc-tangent of the argument. **expression** can have any value. The result is in radians and is between -PI/2 and PI/2.

Arguments
- **expression**
  Any valid BASIC expression.

Example **>> PRINT ATAN(1)**
0.7854

See also N/A

4-2-30 ATAN2

Type Mathematical function

Syntax **ATAN2(expression1, expression2)**

Description The **ATAN2** function returns the arc-tangent of the non-zero complex number **(expression1, expression2)**, which is equivalent to the angle between a point with coordinate **(expression1, expression2)** and the x-axis. If **expression2 >= 0**, the result is equal to the value of **ATAN(expression1 / expression2)**. The result in radians will be between -PI and PI.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **>> PRINT ATAN2(0,1)**
0.0000

See also N/A

4-2-31 ATYPE

Type Axis parameter

Syntax **ATYPE = value**

Description The **ATYPE** axis parameter indicates the axis type for the axis. The valid values depend on the port the Servo Driver controlling the axis is connected to. See the table below.

AXIS type	ATYPE value
Virtual	0
MECHATROLINK-II Position	40

AXIS type	ATYPE value
MECHATROLINK-II Speed	41
MECHATROLINK-II Torque	42
Flexible axis Stepper Out	43
Encoder Interface In	44
Flexible axis Encoder Out	45
Flexible axis Absolute EnDat	47
Flexible axis Absolute SSI	48
MECHATROLINK-II Inverter	49

The **ATYPE** parameters are set by the system at start-up. For axes controlled by the Servo Drivers connected to the system via MECHATROLINK-II bus, the default **ATYPE** value is 41 (MECHATROLINK-II Speed) for Sigma II Servo Drivers, or 40 (MECHATROLINK-II Position) for JUNMA Servo Drivers. For axes controlled by the Servo Drivers connected to the system via the Encoder Interface, the default **ATYPE** value is 44 (Encoder Interface In).

Arguments N/A

Example **ATYPE AXIS(1) = 45**
This command will set axis 1 as Flexible axis encoder output axis.

See also **AXIS**

4-2-32 AUTORUN

Type Program command

Syntax **AUTORUN**

Description The **AUTORUN** command starts all the programs that have been set to run at start-up.

Note: This command should only be used on the Command Line Terminal.

Arguments N/A

Example No example.

See also **RUNTYPE**

4-2-33 AXIS

Type System command

Syntax **AXIS(axis_number)**

Description The **AXIS** modifier sets the axis for a single motion command or a single axis parameter read/write to a particular axis. **AXIS** is effective only for the command or axis parameter operation. If it is required to change the axis used to a particular axis in every subsequent command, use the **BASE** command instead.

Arguments
- **axis_number**
  Any valid BASIC expression specifying the axis number.

Example **BASE(0)**
PRINT VP_SPEED AXIS(2)

Example **MOVE(300) AXIS(0)**

Example **REP_DIST AXIS(1) = 100**

See also **BACKLASH**

4-2-34 AXIS_ENABLE

Type Axis parameter

Syntax **AXIS_ENABLE = ON/OFF**

Description The **AXIS_ENABLE** axis parameter is used to enable or disable particular axis independently of others. This parameter can be set ON or OFF for each axis individually. The default value on start-up is ON or all axes. The axis will be enables if both AXIS_ENABLE for that axis is ON and **WDOG** is on. For MECHATROLINK-II axes setting AXIS_ENABLE to OFF will disable Servo Driver output to the motor. For Flexible axis Servo axis setting **AXIS_ENABLE** to OFF will force both voltage outputs to 0. For Flexible axis Stepper Out and Encoder Out axes, setting **AXIS_ENABLE** to OFF will block pulses generation on the outputs.

Arguments N/A

Example **AXIS_ENABLE AXIS(3) = OFF**
This command will disable axis 3 independently of other axes in the system.

See also **AXIS**, **DISABLE_GROUP**

4-2-35 AXISSTATUS

Type Axis parameter (read-only)

Syntax **AXISSTATUS**

Description The **AXISSTATUS** axis parameter contains the axis status and is used for the motion error handling of the controller. The axis status consists of status bits, which definitions are shown in the table below.

Bit number	Description	Value	Character
0	-	1	-
1	Following error warning range	2	w
2	Servo Driver communication error	4	a
3	Servo Driver alarm	8	m
4	in forward limit	16	f
5	In reverse limit	32	r
6	Datuming	64	d
7	Feed hold input	128	h
8	Following error exceeds limit	256	e
9	In forward software limit	512	x

Bit number	Description	Value	Character
10	In reverse software limit	1024	y
11	Cancelling move	2048	c
12	Encoder out overspeed	4096	o

Arguments N/A

Example **IF (AXISSTATUS AND 16)>0 THEN PRINT "In forward limit"**

See also **AXIS**, **ERRORMASK**

4-2-36 B_SPLINE

Type Axis command

Syntax **B_SPLINE(type, data_in, number_in, data_out, #expand)**

Description Expands an existing profile stored in the TABLE using the B-Spline mathematical function. The expansion factor is configurable and the B_SPLINE stores expanded profile to another area in the TABLE.
This is ideally used where the source **CAM** profile is too course and needs to be extrapolated into a greater number of points.

Arguments
- **type**
  Reserved for future expansion. Always set this to 1.
- **data_in**
  Location in the TABLE where the source profile is stored.
- **number_in**
  Number of points in the source profile.
- **data_out**
  Location in the TABLE where the expanded profile will be stored.
- **expansion_ratio**
  The expansion ratio, i.e., if the source profile is 100 points and **expansion_ratio** is set to 10 the resulting profile will be 1000 point (100 * 10).

Example
BASE(1)
B_SPLINE(1, 0, 10, 200, 10)
This command expands a 10 point profile in TABLE locations 0 to 9 to a larger 100 points profile starting at TABLE location 200.

See also N/A

4-2-37 BACKLASH

Type Axis command

Syntax **BACKLASH(on/off, distance, speed, accel)**

Description The **BACKLASH** command allows the parameters for the backlash compensation to be loaded. The backlash compensation is achieved as follows:

- An offset move is applied when the motor demand is in one direction.
- The offset move is reversed when the motor demand is in the opposite direction.

These moves are superimposed on the command axis movements.
The backlash compensation is applied after a change in the direction of the **DPOS** parameter. The backlash compensation can be seen in the **TRANS_DPOS** parameter, which is equal to **DPOS** + backlash compensation.

Arguments
- **on/off**
  Either ON or OFF.
- **distance**
  The offset distance, expressed in user units.
- **speed**
  The speed of the compensation move, expressed in user units.
- **accel**
  The acceleration or deceleration rate of the compensation move, expressed in user units.

Example
```
BACKLASH(ON,0.5,10,50) AXIS(0)
BACKLASH(ON,0.4,8,50) AXIS(1)
```
This applies backlash compensation on axes 0 and 1.

See also **DPOS**, **TRANS_DPOS**.

4-2-38 BACKLASH_DIST

Type Axis parameter

Syntax **BACKLASH_DIST**

Description **BACKLASH_DIST** is the amount of backlash compensation that is applied to the axis when **BACKLASH = ON**.

Arguments N/A

Example
```
IF BACKLASH_DIST>100 THEN
  OP (10, ON) ' show that backlash compensation reached this value
ELSE
 OP (10, OFF)
END IF
```

See also **BACKLASH**

4-2-39 BASE

Type Axis command

Syntax
```
BASE
BASE(axis_1 [ ,axis_2 [ , axis_3 [ , axis_4 [ , axis_...]]]])
BA
BA(axis_1 [ ,axis_2 [ , axis_3 [ , axis_4 [ , axis_...]]]])
```

Description

The **BASE** command is used to set the default base axis or to set a specified axis sequence group. All subsequent motion commands and axis parameters will apply to the base axis or the specified axis group unless the **AXIS** command is used to specify a temporary base axis. The base axis or axis group is effective until it is changed again with **BASE**.

Each BASIC process can have its own axis group and each program can set its own axis group independently. Use the **PROC** modifier to access the parameters for a certain task.

The **BASE** order grouping can be set by explicitly assigning the order of axes. This order is used for interpolation purposes in multi-axes linear and circular moves. The default for the base axis group is (0,1,2,..30) at start-up or when a program starts running on a task. The **BASE** command without any arguments returns the current base order grouping. This should be used

Note: If the **BASE** command does not specify all the axes, the **BASE** command will “fill in” the remaining values automatically. Firstly it will fill in any remaining axes above the last declared value, then it will fill in any remaining axes in sequence.

So **BASE(2,6,10)** sets the internal array of 16 axes to: 2,6,10,11,12,13,14,15,0,1,3,4,5,7,8,9.

Note: The **BASE** command without any arguments should only be used on the Command Line Terminal.

Arguments

The command can take up to 16 arguments.

- **axis_i**
  The number of the axis set as the base axis and any subsequent axes in the group order for multi-axis moves.

Example

```
BASE(1)
UNITS = 2000 ' Set unit conversion factor for axis 1
SPEED = 100 ' Set speed for axis 1
ACCEL = 5000 ' Set acceleration rate for axis 1
BASE(2)
UNITS = 2000 ' Set unit conversion factor for axis 2
SPEED = 125 ' Set speed for axis 2
ACCEL = 10000 ' Set acceleration rate for axis 2
```

It is possible to program each axis with its own speed, acceleration and other parameters.

Example

```
BASE(0)
MOVE(100,-23.1,1250)
```

In this example, axes 0, 1 and 2 will move to the specified positions at the speed and acceleration set for axis 0. **BASE(0)** sets the base axis to axis 0, which determines the three axes used by **MOVE** and the speed and acceleration rate.

Example

```
>> BASE
(0,2,1)
```

On the command line the base group order can be shown by typing **BASE**.

Example

```
>> RUN "PROGRAM", 3
>> BASE PROC(3)
(0,2,1)
```

Use the **PROC** modifier to show the base group order of a certain task.

Example **>> BASE(2)**
>> PRINT BASE
2.0000
Printing **BASE** will return the current selected base axis.

See also **AXIS**

4-2-40 BASICERROR

Type System command

Syntax **BASICERROR**

Description The **BASICERROR** command can be used to run a routine when a run-time error occurs in a program. **BASICERROR** can only be used as part of an **ON ... GOSUB** or **ON ... GOTO** command. This command is required to be executed once in the BASIC program. If several commands are used only the one executed last is effective.

Arguments N/A

Example
```
 ON BASICERROR GOTO error_routine
 ...
 no_error = 1
 STOP
error_routine:
 IF no_error = 0 THEN
  PRINT "The error "; RUN_ERROR[0];
  PRINT " occurred in line "; ERROR_LINE[0]
 ENDIF
 STOP
```
If an error occurs in a BASIC program in this example, the error routine will be executed.
The **IF** statement is present to prevent the program going into error routine when it is stopped normally.

See also **ERROR_LINE**, **ON**, **RUN_ERROR**.

4-2-41 BATTERY_LOW

Type System parameter (read-only)

Syntax **BATTERY_LOW**

Description This parameter returns the current state of the battery condition. If **BATTERY_LOW=ON** then the battery needs to be changed. If **BATTERY_LOW=OFF** then battery condition is ok.

Arguments N/A

Example No example.

See also N/A

4-2-42 BREAK_RESET

Type System command

Syntax **BREAK_RESET "program_name"**

Description Used by Trajexia Studio to remove all break points from the specified program.

Arguments
- **program_name**
  The name of the program from which you want to remove all break points.

Example **BREAK_RESET "simpletest"**
Will remove all break points from program **simpletest**.

See also N/A

4-2-43 CAM

Type Axis command

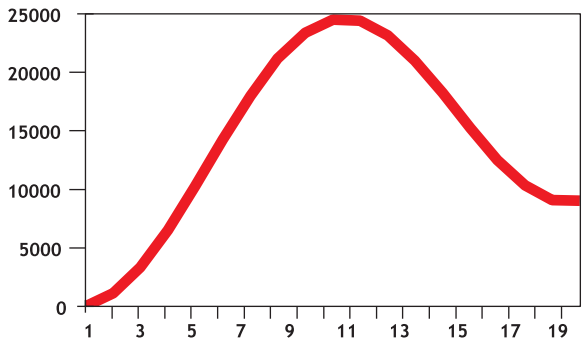
Syntax **CAM(start_point, end_point, table_multiplier, distance)**

Description The **CAM** command is used to generate movement of an axis following a position profile which is stored in the TABLE variable array. The TABLE values are absolute positions relative to the starting point and are specified in encoder edges. The TABLE array is specified with the **TABLE** command.
The movement can be defined with any number of points from 3 to the maximum table size available (64000). The CJ1W-MCH72 moves continuously between the values in the TABLE to allow a number of points to define a smooth profile. Two or more **CAM** commands can be executed simultaneously using the same or overlapping values in the TABLE array. The TABLE profile is traversed once.
CAM requires that the start element in the TABLE array has value zero. The distance argument together with the **SPEED** and **ACCEL** parameters determine the speed moving through the TABLE array. Note that in order to follow the **CAM** profile exactly the **ACCEL** parameter of the axis must be at least 1000 times larger than the **SPEED** parameter.
CAM works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

Arguments

- **start_point**
  The address of the first element in the TABLE array to be used. Being able to specify the start point allows the TABLE array to hold more than one profile and/or other information.
- **end_point**
  The address of the end element in the TABLE array.
- **table_multiplier**
  The Table multiplier value used to scale the values stored in the TABLE. As the Table values are specified in encoder edges, use this argument to set the values for instance to the unit conversion factor (set by **UNITS** parameter).
- **distance**
  A factor given in user units that controls the speed of movement through the Table. The time taken to execute **CAM** depends on the current axis speed and this distance. For example, assume the system is being programmed in mm and the speed is set to 10 mm/s and the acceleration sufficiently high. If a distance of 100 mm is specified, **CAM** will take 10 seconds to execute.
  The **SPEED** parameter in the base axis allows modification of the speed of movement when using the **CAM** move.

Note: When the **CAM** command is executing, the **ENDMOVE** parameter is set to the end of the previous move.



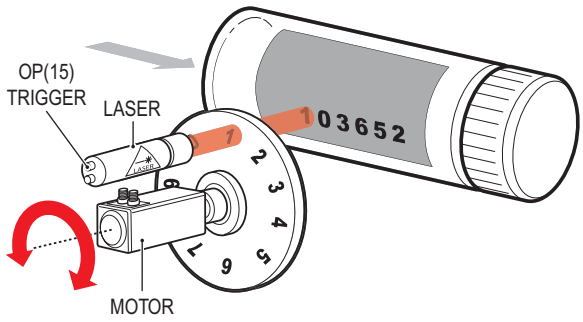
Example

Motion is required to follow the POSITION equation:
t(x) = x*25 + 10000(1-cos(x)), where x is in degrees. This example table provides a simple oscillation superimposed with a constant speed. To load the table and cycle it continuously the program would be:

```
FOR deg=0 TO 360 STEP 20 'loop to fill in the table
  rad = deg * 2 * PI/360 'convert degrees to radians
  x = deg * 25 + 10000 * (1-COS(rad))
  TABLE(deg/20,x) 'place value of x in table
NEXT deg
WHILE IN(2)=ON 'repeat cam motion while input 2 is on
  CAM(0,18,1,200)
  WAIT IDLE
WEND
```

Note: The subroutine **camtable** loads the data into the cam TABLE, as shown in the figure and in the table below.

TABLE position	Degree	Value
1	0	0
2	20	1103
3	40	3340
4	60	6500
5	80	10263
6	100	14236
7	120	18000
8	140	21160
9	160	23396
10	180	24500
11	200	24396
12	220	23160
13	240	21000
14	260	18236
15	280	15263
16	300	12500
17	320	10340
18	340	9103
19	360	9000



Example A masked wheel is used to create a stencil for a laser to shine through for use in a printing system for the ten numerical digits. The required digits are transmitted through port 1 serial port to the controller as ASCII text.

The encoder used has 4000 edges per revolution and so must move 400 between each position. The cam table goes from 0 to 1, which means that the **CAM** multiplier needs to be a multiple of 400 to move between the positions.

The wheel is required to move to the pre-set positions every 0.25 seconds. The speed is set to 10000 edges/second, and we want the profile to be complete in 0.25 seconds. So multiplying the axis speed by the required completion time (10000 x 0.25) gives the distance parameter equals 2500.

```
GOSUB profile_gen
WHILE IN(2)=ON
  WAIT UNTIL KEY#1 'Waits for character on port 1
  GET#1,k
  IF k>47 AND k<58 THEN 'check for valid ASCII character
    position=(k-48)*400 'convert to absolute position
    multiplier=position-offset 'calculate relative movement
    'check if it is shorter to move in reverse direction
    IF multiplier>2000 THEN
      multiplier=multiplier-4000
    ELSEIF multiplier<-2000 THEN
      multiplier=multiplier+4000
    ENDIF
    CAM(0,200,multiplier,2500) 'set the CAM movment
    WAIT IDLE
    OP(15,ON) 'trigger the laser flash
    WA(20)
    OP(15,OFF)
    offset=(k-48)*400 'calculates current absolute position
  ENDIF
WEND
```

```
profile_gen:
  num_p=201
  scale=1.0
  FOR p=0 TO num_p-1
    TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
  NEXT p
RETURN
```

Example

A suction pick and place system must vary its speed depending on the load carried. The mechanism has a load cell which inputs to the controller on the analogue channel (**AIN**).
The move profile is fixed, but the time taken to complete this move must be varied depending on the **AIN**. The **AIN** value varies from 100 to 800, which must result in a move time of 1 to 8 seconds. If the speed is set to 10000 units per second and the required time is 1 to 8 seconds, then the distance parameter must range from 10000 to 80000. (distance = speed x time).
The return trip can be completed in 0.5 seconds and so the distance value of 5000 is fixed for the return movement. The Multiplier is set to -1 to reverse the motion.

```
GOSUB profile_gen 'loads the cam profile into the table
SPEED=10000:ACCEL=SPEED*1000:DECEL=SPEED*1000
WHILE IN(2)=ON
  OP(15,ON) 'turn on suction
  load=AIN(0) 'capture load value
  distance = 100*load 'calculate the distance parameter
  CAM(0,200,50,distance) 'move 50mm forward in time calculated
  WAIT IDLE
  OP(15,OFF) 'turn off suction
  WA(100)
  CAM(0,200,-50,5000) 'move back to pick up position
WEND

profile_gen:
  num_p=201
  scale=400 'set scale so that multiplier is in mm
  FOR p=0 TO num_p-1
    TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
  NEXT p
RETURN
```

See also **ACCEL**, **AXIS**, **CAMBOX**, **SPEED**, **TABLE**.

4-2-44 CAMBOX

Type Axis command

Syntax **CAMBOX(start_point, end_point, table_multiplier, link_distance, link_axis [, link_option [, link_position]])**

Description

The **CAMBOX** command is used to generate movement of an axis following a position profile in the TABLE variable array. The motion is linked to the measured motion of another axis to form a continuously variable software gearbox. The TABLE values are absolute position relative to the starting point and are specified in encoder edges.

The TABLE array is specified with the **TABLE** command. The movement can be defined with any number of points from 3 to the maximum table size available (64000). Being able to specify the start point allows the TABLE array to be used to hold more than one profile and/or other information. The CJ1W-MCH72 moves continuously between the values in the TABLE to allow a number of points to define a smooth profile. Two or more **CAMBOX** commands can be executed simultaneously using the same or overlapping values in the TABLE array.

The **CAMBOX** command requires the start element of the TABLE to have value zero. Note also that **CAMBOX** command allows traversing the TABLE backwards as well as forwards depending on the Master axis direction.

The **link_option** argument can be used to specify different options to start the command and to specify a continuous **CAM**. For example, if the **link_option** is set to 4 then the **CAMBOX** operates like a "physical" **CAM**.

CAMBOX works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

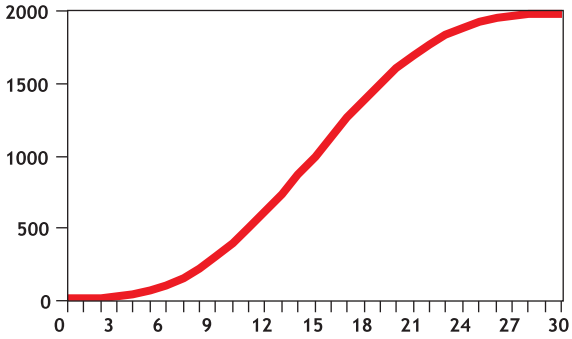
Note: While **CAMBOX** is being executed, the **ENDMOVE** parameter will be set to the end of the previous move. The **REMAIN** axis parameter will hold the remainder of the distance on the link axis.

Arguments

- **start_point**
  The address of the first element in the TABLE array to be used.
- **end_point**
  The address of the end element in the TABLE array.
- **table_multiplier**
  The Table multiplier value used to scale the values stored in the TABLE. As the TABLE values are specified in encoder edges, use this argument to set the values for instance to the unit conversion factor (set by UNITS parameter).
- **link_distance**
  The distance in user units the link axis must move to complete the specified output movement. The link distance must be specified as a positive distance.
- **link_axis**
  The axis to link to.
- **link_option**
  See the table below.

link_option value	Description
1	Link starts when registration event occurs on link axis.
2	Link starts at an absolute position on link axis (see **link_position**).
4	**CAMBOX** repeats automatically and bidirectionally. This option is cancelled by setting bit 1 of **REP_OPTION** parameter (**REP_OPTION = REP_OPTION OR 2**).
5	Combination of options 1 and 4.
6	Combination of options 2 and 4.

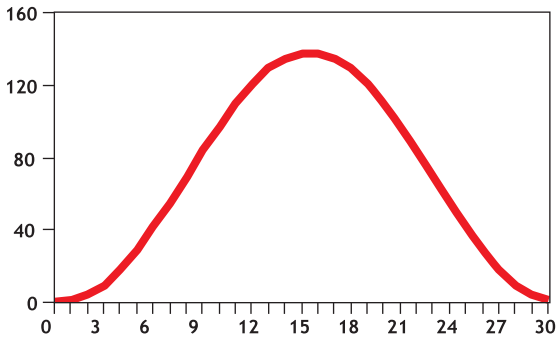
- **link_position**
  The absolute position where **CAMBOX** will start when **link_option** is set to 2.
  Note: When the **CAMBOX** command is executing, the **ENDMOVE** parameter is set to the end of the previous move. The **REMAIN** axis parameter holds the remainder of the distance on the link axis.

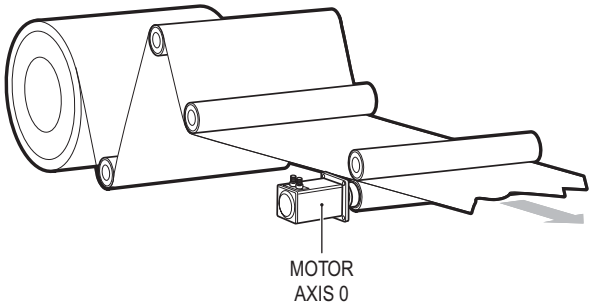


Example

```
' Subroutine to generate a SIN shape speed profile
' Uses: p is loop counter
' num_p is number of points stored in tables pos 0..num_p
' scale is distance travelled scale factor
profile_gen:
  num_p=30
  scale=2000
  FOR p=0 TO num_p
    TABLE(p,((-SIN(PI*2*p/num_p)/(PI*2))+p/num_p)*scale)
  NEXT p
RETURN
```

This graph plots TABLE contents against table array position. This corresponds to motor POSITION against link POSITION when called using **CAMBOX**. The **SPEED** of the motor will correspond to the derivative of the position curve above.



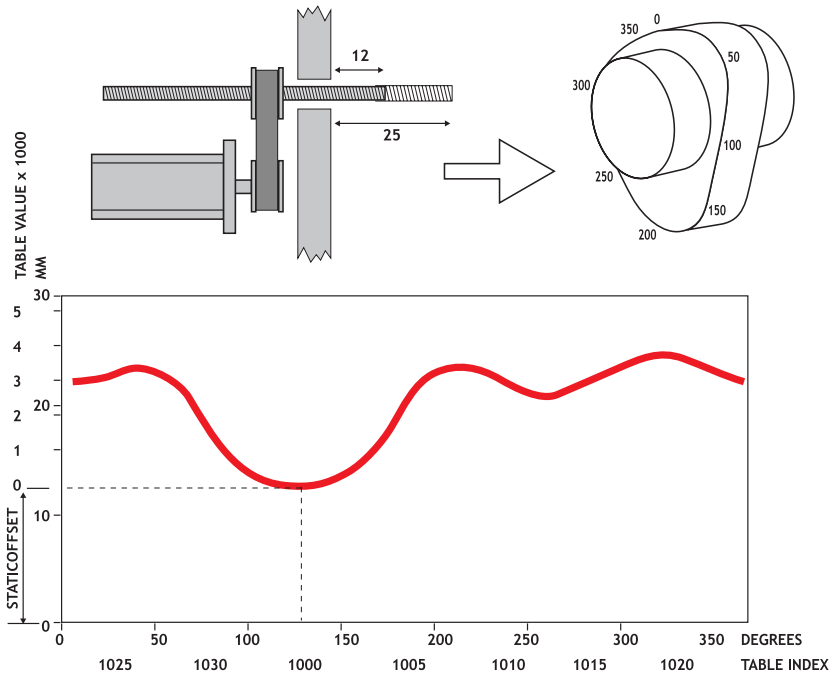


Example A pair of rollers feeds plastic film into a machine. The feed is synchronised to a master encoder and is activated when the master reaches a position held in the variable **start**. This example uses the table points 0...30 generated in the example above:

```
start=1000
FORWARD AXIS(1)
WHILE IN(2)=OFF
  CAMBOX(0,30,800,80,15,2,start)
  WA(10)
  WAIT UNTIL MTYPE=0 OR IN(2)=ON
WEND
CANCEL
CANCEL AXIS(1)
WAIT IDLE
```

The arguments of the **CAMBOX** command are:

- 0 is the start of the profile shape in the TABLE
- 30 is the end of the profile shape in the TABLE
- 800 scales the TABLE values. Each **CAMBOX** motion therefore totals 800*2000 encoder edges steps.
- 80 is the distance on the product conveyor to link the motion to. The units for this parameter are the programmed distance units on the link axis.
- 15 specifies the axis to link to.
- 2 is the link option setting. It means: Start at absolute position on the link axis.
- The variable **start** holds a position. The motion will execute when this position is reached on axis 15.



Example

A motor on Axis 0 is required to emulate a rotating mechanical CAM. The position is linked to motion on axis 3. The “shape” of the motion profile is held in TABLE values 1000..1035.

The table values represent the mechanical cam but are scaled to range from 0-4000.

```
TABLE(1000,0,0,167,500,999,1665,2664,3330,3497,3497)
TABLE(1010,3164,2914,2830,2831,2997,3164,3596,3830,3996,3996)
TABLE(1020,3830,3497,3330,3164,3164,3164,3330,3467,3467,3164)
TABLE(1030,2831,1998,1166,666,333,0)
BASE(3)
MOVEABS(130)
WAIT IDLE
'start the continuously repeating cambox
CAMBOX(1000,1035,1,360,3,4) AXIS(0)
FORWARD start camshaft axis
WAIT UNTIL IN(2)=OFF
REP_OPTION = 2 'cancel repeating mode by setting bit 1
WAIT IDLE AXIS(0) waits for cam cycle to finish
CANCEL 'stop camshaft axis
WAIT IDLE
```

Note: The system software resets bit 1 of **REP_OPTION** after the repeating mode has been cancelled.

Setting bit 3 (value 8) of the link options parameter enables the **CAMBOX** pattern mode. This mode enables a sequence of scale values to be cycled automatically. This is normally combined with the automatic repeat mode, so the options parameter must be set to 12. This diagram shows a typical repeating pattern which can be automated with the **CAMBOX** pattern mode.

The parameters for this mode are treated differently to the standard **CAMBOX** function:

CAMBOX(start, end, control block pointer, link dist, link axis,options)

The start and end parameters specify the basic shape profile ONLY. The pattern sequence is specified in a separate section of the TABLE memory. There is a new TABLE block defined: The “Control Block”. This block of seven TABLE values defines the pattern position, repeat controls etc. The block is fixed at 7 values long.

Therefore in this mode only there are 3 independently positioned TABLE blocks used to define the required motion:

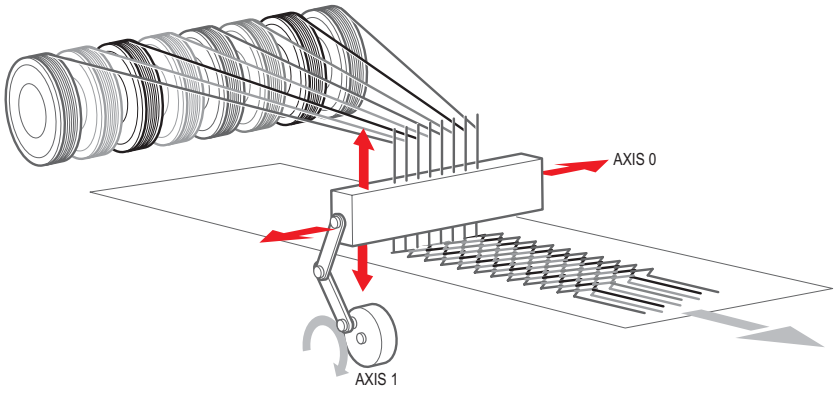
- SHAPE BLOCK: This is directly pointed to by the **CAMBOX** command as in any **CAMBOX**.
- CONTROL BLOCK: This is pointed to by the third **CAMBOX** parameter in this options mode only. It is of fixed length (7 table values). It is important to note that the control block is modified during the **CAMBOX** operation. It must therefore be re-initialised prior to each use.
- PATTERN BLOCK: The start and end of this are pointed to by 2 of the CONTROL BLOCK values. The pattern sequence is a sequence of scale factors for the SHAPE.

The table below gives the CONTROL BLOCK parameters

Note: READ/WRITE values can be written to by the user program during the pattern **CAMBOX** execution.

Value	Parameter	R/W	Description
0	CURRENT POSITION	R	The current position within the TABLE of the pattern sequence. This value should be initialised to the START PATTERN number.
1	FORCE POSITION	R/W	Normally this value is -1. If at the end of a SHAPE the user program has written a value into this TABLE position the pattern will continue at this position. The system software will then write -1 into this position. The value written must be inside the pattern such that the value: CB(2)<=CB(1)<=CB(3)
2	START PATTERN	R	The position in the TABLE of the first pattern value.
3	END PATTERN	R	The position in the TABLE of the final pattern value.
4	REPEAT POSITION	R/W	The current pattern repeat number. Initialise this number to 0. The number will increment when the pattern repeats if the link axis motion is in a positive direction. The number will decrement when the pattern repeats if the link axis motion is in a negative direction. Note that the counter runs starting at zero: 0,1,2,3...

Value	Parameter	R/W	Description
5	REPEAT COUNT	R/W	Required number of pattern repeats. If -1 the pattern repeats endlessly. The number should be positive. When the ABSOLUTE value of CB(4) reaches CB(5) the CAMBOX finishes if CB(6)=-1. The value can be set to 0 to terminate the **CAMBOX** at the end of the current pattern. The axis the CAMBOX is linked to can run in a positive or negative direction. In the case of a negative direction link the pattern will execute in reverse. In the case where a certain number of pattern repeats is specified with a negative direction link, the first control block will produce one repeat less than expected. This is because the CAMBOX loads a zero link position which immediately goes negative on the next servo cycle triggering a REPEAT COUNT. This effect only occurs when the CAMBOX is loaded, not on transitions from CONTROL BLOCK to CONTROL BLOCK. This effect can easily be compensated for either by increasing the required number of repeats, or setting the initial value of REPEAT POSITION to 1.
6	NEXT CONTROL BLOCK	R/W	If set to -1 the pattern will finish when the required number of repeats are done. Alternatively a new control block pointer can be used to point to a further control block.



Example A quilt stitching machine runs a feed cycle that stitches a plain pattern before it starts a patterned stitch. The plain pattern must run for 1000 cycles. Then, it must runs a pattern continuously, until requested to stop at the end of the pattern. The cam profile controls the motion of the needle bar between moves. The pattern table controls the distance of the move to make the pattern.
The same shape is used for the initialisation cycles and the pattern. This shape is held in TABLE values 100..150. The running pattern sequence is held in TABLE values 1000..4999. The initialisation pattern is a single value held in TABLE(160). The initialisation control block is held in TABLE(200)..TABLE(206). The running control block is held in TABLE(300)..TABLE(306).

```
' Set up Initialisation control block:
TABLE(200,160,-1,160,160,0,1000,300)
' Set up running control block:
TABLE(300,1000,-1,1000,4999,0,-1,-1)
' Run whole lot with single CAMBOX:
' Third parameter is pointer to first control block
CAMBOX(100,150,200,5000,1,20)
WAIT UNTIL IN(7)=OFF
TABLE(305,0) ' Set zero repeats: This will stop at end of pattern
```

Note: The axis to which the CAMBOX is linked can run in a positive or negative direction. In the case of a negative direction link, the pattern executes in reverse. In the case where a certain number of pattern repeats is specified with a negative direction link, the first control block produces one repeat less than expected. This is because the CAMBOX loads a zero link position which immediately goes negative on the next servo cycle triggering a REPEAT COUNT. This effect only occurs when the CAMBOX is loaded, not on transitions from CONTROL BLOCK to CONTROL BLOCK. This effect can easily be compensated for: either increase the required number of repeats, or set the initial value of REPEAT POSITION to 1.

See also **AXIS**, **CAM**, **REP_OPTION**, **TABLE**

4-2-45 CANCEL

Type Axis command

Syntax **CANCEL[(1)]**
CA[(1)]

Description

The **CANCEL** command cancels the move on an axis or an interpolating axis group. Speed-profiled moves (**FORWARD**, **REVERSE**, **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL** and **MOVEMODIFY**) will be decelerated at the deceleration rate as set by the **DECEL** parameter and then stopped. Other moves will be immediately stopped.

The **CANCEL** command cancels the contents of the current move buffer (**MTYPE**). The command **CANCEL(1)** cancels the contents of the next move buffer (**NTYPE**) without affecting the current move in the **MTYPE** buffer.

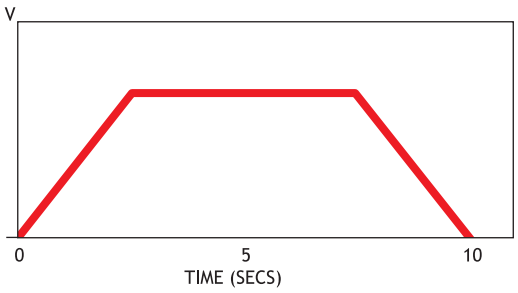
CANCEL works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

Note:

- **CANCEL** cancels only the presently executing move. If further moves are buffered they will then be loaded.
- During the deceleration of the current move additional **CANCEL**s will be ignored.
- **CANCEL(1)** cancels only the presently buffered move. Any moves stored in the task buffers indicated by the **PMOVE** variable can be loaded into the buffer as soon as the buffered move is cancelled.

Arguments

N/A



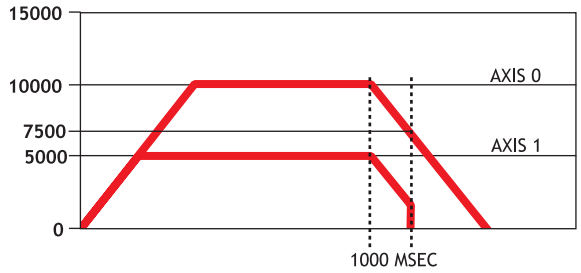
Example

```
FORWARD
WA(10000)
CANCEL  ' Stop movement after 10 seconds
```

Example

```
MOVE(1000)
MOVEABS(3000)
CANCEL ' Cancel the move to 3000 and move to 4000 instead.
MOVEABS(4000)
```

Note that the command **MOVEMODIFY** is a better solution for modifying end points of moves in this case.



Example Two axes are connected with a ratio of 1:2. Axis 0 is cancelled after 1 second, then axis 1 is cancelled when the speed drops to a specified level. After the first cancel axis 1 decelerates at the **DECEL** rate. When the **CONNECT** of axis 1 is cancelled, axis 1 stops instantly.

```
BASE(0)
SPEED=10000
FORWARD
CONNECT(0.5,0) AXIS(1)
WA(1000)
CANCEL
WAIT UNTIL VP_SPEED<=7500
CANCEL AXIS(1)
```

See also **AXIS**, **MTYPE**, **NTYPE**, **PMOVE**, **RAPIDSTOP**

4-2-46 CHECKSUM

Type System parameter (read-only)

Syntax **CHECKSUM**

Description The **CHECKSUM** parameter contains the checksum for the programs in RAM. At start-up, the checksum is recalculated and compared with the previously held value. If the checksum is incorrect the program will not run.

Arguments N/A

Example No example.

See also N/A

4-2-47 CHR

Type I/O command

Syntax **CHR(x)**

Description The **CHR** command is used to send individual ASCII characters which are referred to by number. **PRINT CHR(x)**; is equivalent to **PUT(x)** in some other versions of BASIC.

Arguments
- **x**
  A BASIC expression.

Example
```
>>PRINT CHR(65);
A
```

See also N/A

4-2-48 CLEAR

Type System command

Syntax **CLEAR**

Description The **CLEAR** command resets all global VR variables to 0 and sets local variables on the process on which the command is run to 0. When you use it in a program it resets all local variables defined to 0.

Arguments N/A

Example
```
>>VR(0)=22: VR(20)=44.3158: VR(300)=-12
>>PRINT VR(0), VR(20), VR(300)
22.0000  44.3158  -12.0000
>>CLEAR
>>PRINT VR(0), VR(20), VR(300)
0.0000  0.0000  0.0000
```

See also
- **RESET**, **VR**

4-2-49 CLEAR_BIT

Type System command

Syntax **CLEAR_BIT(bit_number, vr_number)**

Description The **CLEAR_BIT** command resets the specified bit in the specified VR variable. Other bits in the variable keep their values.

Arguments
- **bit_number**
  The number of the bit to be reset. Range: 0 - 23.
- **vr_number**
  The number of the VR variable for which the bit will be reset. Range: 0 - 1023.

Example
```
>>PRINT VR(17)
112.0000
>>CLEAR_BIT(5, 17)
>>PRINT VR(17)
80.0000
```

See also **READ_BIT**, **SET_BIT**, **VR**.

4-2-50 CLEAR_PARAMS

Type System command

Syntax **CLEAR_PARAMS**

Description Clears all variables and parameters stored in flash EPROM to their default values. The CLEAR_PARAM will erase (set to 0) all the VRs stored using FLASHVR command. This command cannot be performed if the controller is locked.

Arguments N/A

Example No example.

See also N/A

4-2-51 CLOSE_WIN

Type Axis parameter

Syntax **CLOSE_WIN**
CW

Description The **CLOSE_WIN** axis parameter defines the end of the window inside or outside which a registration mark is expected. The value is in user units.

Arguments N/A

Example **CLOSE_WIN**=10

See also **AXIS**, **OPEN_WIN**, **REGIST**, **UNITS**.

4-2-52 CLUTCH_RATE

Type Axis parameter

Syntax **CLUTCH_RATE**

Description The **CLUTCH_RATE** axis parameter defines the change in connection ratio when using the **CONNECT** command. The rate is defined as amount of ratio per second.
The default value is set to a high value (1000000) in order to ensure compatibility with other units.
Note: The operation using **CLUTCH_RATE** is not deterministic in position. If required, use the **MOVELINK** command instead to avoid unnecessary phase difference between base axis and linked axis.

Arguments N/A

Example **CLUTCH_RATE = 4**
This setting will imply that when giving **CONNECT(4,1)**, it will take one second to reach the full connection.

See also **AXIS**, **CONNECT**, **MOVELINK**.

4-2-53 COMPILE

Type Program command

Syntax **COMPILE**

Description The **COMPILE** command forces the compilation of the currently selected program to intermediate code. Program are compiled automatically by the system software prior to program execution or when another program is selected. This command is not therefore normally required.

Arguments N/A

Example No example.

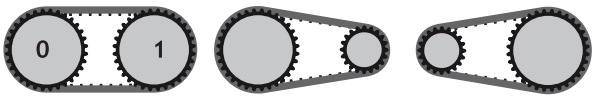
See also N/A

4-2-54 CONNECT

Type Axis command

Syntax **CONNECT(ratio, driving_axis)**
CO(ratio, driving_axis)

Description The **CONNECT** command connects the demand position of the base axis to the measured movements of the axis specified by **driving_axis** to achieve an electronic gearbox.
The ratio can be changed at any time by executing another **CONNECT** command on the same axis. To change the driving axis the **CONNECT** command needs to be cancelled first. **CONNECT** with different driving axis will be ignored. The **CONNECT** command can be cancelled with a **CANCEL** or **RAPIDSTOP** command. The **CLUTCH_RATE** axis parameter can be used to set a specified connection change rate.
CONNECT works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

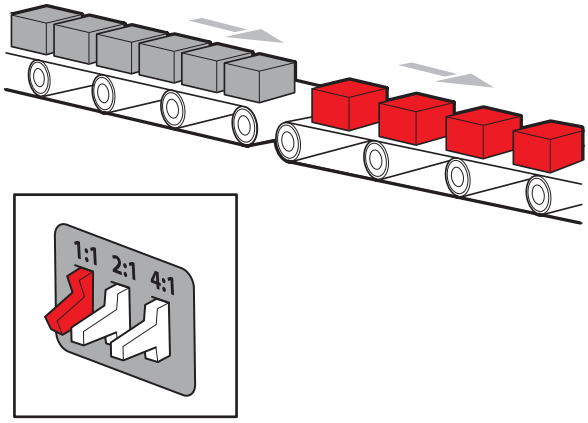


CONNECT (1,1) CONNECT (2,1) CONNECT (0.5,1)

Arguments
- **ratio**
  The connection ratio of the gearbox. The ratio is specified as the encoder edge ratio (not units). It holds the number of edges the base axis is required to move per edge increment of the driving axis. The ratio value can be either positive or negative and has sixteen bit fractional resolution.
- **driving_axis**
  The Master axis which will drive the base axis.
  Note: To achieve an exact connection of fractional ratio's of values such as 1024/3072 the **MOVELINK** command can be used with the continuous repeat link option set to ON.

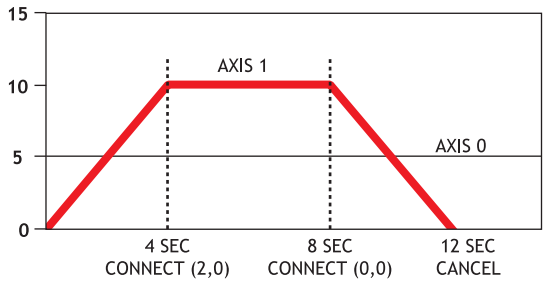
Example In a press feed, a roller is required to rotate at a speed that is equal to one quarter of the measured rate from an encoder installed on the incoming conveyor. The roller is wired to the master axis 0. The reference encoder is connected to axis 1.

```
BASE(0)
SERVO=ON
CONNECT(0.25,1)
```



Example A machine has an automatic feed on axis 1 that must move at a set ratio to axis 0. This ratio is selected using inputs 0-2 to select a particular “gear”. This ratio can be updated every 100 ms. Combinations of inputs select the intermediate gear ratios. For example, 1 ON and 2 ON gives a ratio of 6:1.

```
BASE(1)
FORWARD AXIS(0)
WHILE IN(3)=ON
  WA(100)
  gear = IN(0,2)
  CONNECT(gear,0)
WEND
RAPIDSTOP  cancel the FORWARD and the CONNECT
```



Example Axis 0 is required to run a continuous forward. Axis 1 must connect to axis 0. If **CONNECT** is called, it results in a step change. Therefore, **CLUTCH_RATE** is used, together with an initial and final connect ratio of zero, to get the required motion.

```
FORWARD AXIS(0)
BASE(1)
CONNECT(0,0) 'set intitial ratio to zero
CLUTCH_RATE=0.5 'set clutch rate
CONNECT(2,0) 'apply the required connect ratio
WA(8000)
CONNECT(0,0) 'apply zero ratio to disconnect
WA(4000) 'wait for deceleration to complete
CANCEL 'cancel connect
```

See also **AXIS**, **CANCEL**, **CLUTCH_RATE**, **CONNECT**, **RAPIDSTOP**.

4-2-55 CONSTANT

Type System command

Syntax **CONSTANT "name", value**

Description Declares the name as a constant for use both within the program containing the **CONSTANT** definition and all other programs in the Trajexia Studio solution.

Note: The program containing the **CONSTANT** definition must be run before the name is used in other programs. In addition, only that program should be running at the time the CONSTANT is executed, otherwise the program error will appear and the program will stop when trying to execute this command. For fast startup the program should also be the only process running at power-up.
When the CONSTANT is declared, the declaration remains active until the next CJ1W-MCH72 reset by switching the power off and back on, or by executing the EX command.
A maximum of 128 **CONSTANT**s can be declared.

Arguments
- **name**
  Any user-defined name containing lower case alpha, numerical or underscore characters.
- **value**
  The value assigned to **name**.

Example

```
CONSTANT "nak", $15
CONSTANT "start_button", 5
IF IN(start_button)=ON THEN OP(led1, ON)
IF key_char=nak THEN GOSUB no_ack_received
```

See also N/A

4-2-56 CONTROL

Type System parameter (read-only)

Syntax **CONTROL**

Description The **CONTROL** parameter returns the type of controller in the system. The value of this system parameter for the CJ1W-MCH72 is 264.

Note: When the Motion Controller is locked, 1000 is added to the value, so a locked CJ1W-MCH72 will return 1264.

Arguments N/A

Example No example.

See also N/A

4-2-57 COPY

Type Program command

Syntax **COPY program_name new_program_name**

Description The **COPY** command copies an existing program in the controller to a new program with the specified name. The program name can be specified without quotes.
Note: This command is implemented for the Command Line Terminal.

Arguments
- **program_name**
  Name of the program to be copied.
- **new_program_name**
  Name to use for the new program.

Example **>> COPY "prog" "newprog"**

See also **DEL**, **NEW**, **RENAME**.

4-2-58 COS

Type Mathematical function

Syntax **COS(expression)**

Description The **COS** function returns the cosine of the expression. Input values are in radians and may have any value. The result value will be in the range from -1 to 1.

Arguments
- **expression**
  Any valid BASIC expression.

Example **>> PRINT COS(0)**
1.0000

See also N/A

4-2-59 CREEP

Type Axis parameter

Syntax **CREEP**

Description The **CREEP** axis parameter contains the creep speed for the axis. The creep speed is used for the slow part of an origin search sequence. **CREEP** can have any positive value, including 0.
The creep speed is entered in units/sec with the unit conversion factor **UNITS**. For example, if the unit conversion factor is set to the number of encoder edges/inch, the speed is set in inches/sec.

Arguments N/A

Example
```
BASE(2)
CREEP=10
SPEED=500
DATUM(4)
CREEP AXIS(1)=10
SPEED AXIS(1)=500
DATUM(4) AXIS(1)
```

See also **AXIS**, **DATUM**, **UNITS**.

4-2-60 D_GAIN

Type Axis parameter

Syntax **D_GAIN**

Description The **D_GAIN** axis parameter contains the derivative gain for the axis. The derivative output contribution is calculated by multiplying the change in Following Error with **D_GAIN**. The default value is 0.
Add the derivative gain to a system to produce a smoother response and to allow the use of a higher proportional gain that could not be used otherwise. High values can cause oscillation.
Note: The servo gain must only be changed when the **SERVO** is off.
Note: Servo gains have no affect on stepper output axis, ATYPE=46.

Arguments N/A

Example D_GAIN=0.25

See also
- **AXIS**, **I_GAIN**, **OV_GAIN**, **P_GAIN**, **VFF_GAIN**.

4-2-61 D_ZONE_MAX

Type System parameter

Syntax **D_ZONE_MAX=value**

Description This parameter works in conjunction with **D_ZONE_MIN** to clamp the S_REF output to zero when the demand movement is complete and the magnitude of the Following Error is less than the **D_ZONE_MIN** value. The servo loop will be reactivated when either the Following Error rises above the **D_ZONE_MAX** value, or a fresh movement is started.

Arguments N/A

Example
```
D_ZONE_MIN=3
D_ZONE_MAX=10
```
With these 2 parameters set as above, the S_REF output will be clamped at zero when the movement is complete and the Following Error falls below 3. When a movement is restarted or if the Following Error rises above a value of 10, the servo loop will be reactivated.

See also **D_ZONE_MIN**.

4-2-62 D_ZONE_MIN

Type System parameter

Syntax **D_ZONE_MIN=value**

Description This parameter works in conjunction with **D_ZONE_MAX** to clamp the S_REF output to zero when the demand movement is complete and the magnitude of the Following Error is less than the **D_ZONE_MIN** value. The servo loop will be reactivated when either the Following Error rises above the **D_ZONE_MAX** value, or a fresh movement is started.

Arguments N/A

Example **D_ZONE_MIN=3**
D_ZONE_MAX=10
With these 2 parameters set as above, the S_REF output will be clamped at zero when the movement is complete and the Following Error falls below 3. When a movement is restarted or if the Following Error rises above a value of 10, the servo loop will be reactivated.

See also **D_ZONE_MAX**.

4-2-63 DATE

Type System parameter (read-only)

Syntax **DATE**

Description Returns the current date held by the real time clock of the PLC.

Arguments N/A

Example **>>PRINT DATE**
36956
This prints the number representing the current day. This number is the number of days since 1st January 1900, with 1 Jan. 1900 represented as 1.

See also N/A

4-2-64 DATE$

Type System command

Syntax **DATE$**

Description Prints the current date DD/MM/YY as a string to the communication port. A 2-digit year description is given.

Arguments N/A

Example **PRINT #1, DATE$**
This will print the date in format for example: 20/10/05

See also N/A

4-2-65 DATUM

Type Axis command

Syntax **DATUM(sequence)**

Description

The **DATUM** command performs one of 6 origin search sequences to position an axis to an absolute position and also reset the error bits in **AXISSTATUS** axis parameter.

DATUM uses both the creep and demand speed for the origin search. The creep speed in the sequences is set with the **CREEP** axis parameter and the demand speed is set with the **SPEED** axis parameter. The datum switch input number, used for sequences 3 to 6, is set by the **DATUM_IN** parameter.

DATUM works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

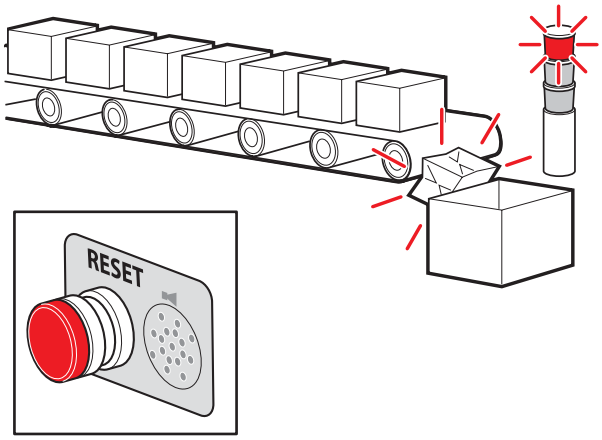
Note: The origin input set with the **DATUM_IN** parameter is active low, i.e., the origin switch is set when the input is OFF. The feedhold, reverse jog, forward jog, forward and reverse limit inputs are also active low. Active low inputs are used to enable fail-safe wiring.

Arguments

- **sequence**
  See the table below.

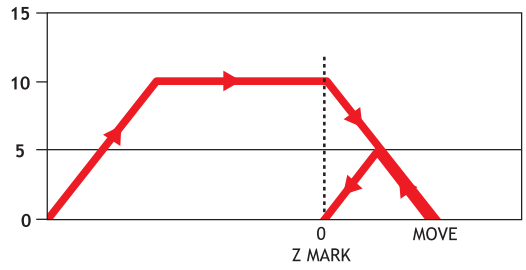
sequence value	Description
0	The **DATUM(0)** command will clear the motion error. The currently measured position is set as the demand position (this is especially useful on stepper axes with position verification). DATUM(0) also clears the Following Error that exceeded the FE_LIMIT condition in the **AXISSTATUS** register for ALL axes. It sets these bits in AXXISSTATUS to zero: Bit 1 : Following Error Warning. Bit 2 : Remote Driver Comms Error. Bit 3 : Remote Driver Error. Bit 8 : Following Error Limit Exceeded. Bit 11 : Cancelling Move. Note that the status can not be cleared if the cause of the problem is still present. **DATUM(0)** must only be used after the WDOG is set to OFF, otherwise there will be unpredictable errors on the motion.
1	The axis moves at creep speed forward until the Z marker is encountered. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.
2	The axis moves at creep speed in reverse until the Z marker is encountered. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.
3	The axis moves at the demand speed forward until the datum switch is reached. The axis then moves reverse at creep speed until the datum switch is reset. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.
4	The axis moves at the demand speed in reverse until the datum switch is reached. The axis then moves forward at creep speed until the datum switch is reset. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.

sequence value	Description
5	The axis moves at demand speed forward until the datum switch is reached. The axis then reverses at creep speed until the datum switch is reset. The axis continues in reverse at creep speed until the Z marker of the encoder is encountered. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.
6	The axis moves at demand speed reverse until the datum switch is reached. The axis then moves forward at creep speed until the datum switch is reset. The axis continues forward at creep speed until the Z marker of the encoder is encountered. The demand position is then reset to 0 and the measured position corrected so as to maintain the Following Error.



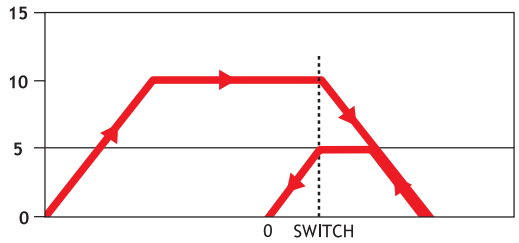
Example A production line must stop if something blocks the product belt, which causes a motion error. The obstacle must be removed, and a reset button must be pressed to restart the line.

```
FORWARD 'start production line
WHILE IN(2)=ON
 IF MOTION_ERROR=0 THEN
  OP(8,ON) 'green light on; line is in motion
 ELSE
  OP(8, OFF)
  GOSUB error_correct
 ENDIF
WEND
CANCEL
STOP

error_correct:
 REPEAT
  OP(10,ON)
  WA(250)
  OP(10,OFF) 'flash red light to show crash
  WA(250)
 UNTIL IN(1)=OFF
 DATUM(0) 'reset axis status errors
 SERVO=ON 'turn the servo back on
 WDOG=ON 'turn on the watchdog
 OP(9,ON) 'sound siren that line will restart
 WA(1000)
 OP(9,OFF)
 FORWARD 'restart motion
RETURN
```



Example The position of an axis must be defined by the Z marker. This position must be set to zero. Then the axis must move to this position. Using the datum 1 the zero point is set on the Z mark. But the axis starts to decelerate at this point, and therefore it stops after the mark. A move is used to bring it back to the Z position.

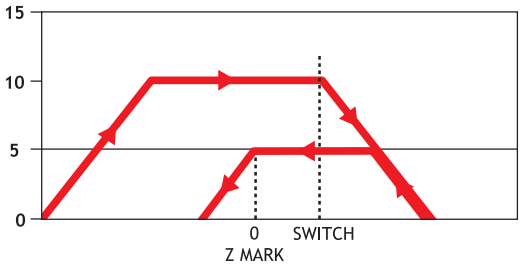
```
SERVO=ON
WDOG=ON
CREEP=1000 'set the search speed
SPEED=5000 'set the return speed
DATUM(1) 'register on Z mark and sets this to datum
WAIT IDLE
MOVEABS (0) 'moves to datum position
```



Example

A machine must return to its home position defined by the limit switch which is found at the rear of the move before operation. This can be achieved through using **DATUM(4)** which moves in reverse to find the switch.

```
SERVO=ON
WDOG=ON
REV_IN=-1 'temporarily turn off the limit switch function
DATUM_IN=5 'sets input 5 for registration
SPEED=5000 'set speed, for quick location of limit switch
CREEP=500 'set creep speed for slow move to find edge of switch
DATUM(4) 'find edge at creep speed and stop
WAIT IDLE
DATUM_IN=-1
REV_IN=5 'restore input 5 as a limit switch again
```



Example A machine similar to the machine in the example above must locate a home switch, which is at the forward end of the move. The machine then moves backwards to the next Z marker, and set this Z marker as the
datum. This is done with **DATUM(5)**, which moves forward at **SPEED** to locate the switch, then reverses at **CREEP** to the Z marker. If required, a move is made to the datum Z marker.

```
SERVO=ON
WDOG=ON
DATUM_IN=7 'sets input 7 as home switch
SPEED=5000 'set speed, for quick location of switch
CREEP=500 'set creep speed for slow move to find edge of switch
DATUM(5) 'start the homing sequence
WAIT IDLE
```

See also **ACCEL**, **AXIS**, **AXISSTATUS**, **CREEP**, **DATUM_IN**, **DECEL**, **MOTION_ERROR**, **SPEED**.

Note The current Trajexia firmware version (1.6652) does not properly support this command for Sigma-V Servo Drivers.

4-2-66 DATUM_IN

Type Axis parameter

Syntax **DATUM_IN**
DAT_IN

Description The **DATUM_IN** axis parameter contains the input number to be used as the datum switch input for the **DATUM** command. The valid input range is given by 0 to 31. Values 0 to 15 represent physically present inputs of CJ1W-MCH72 I/O connector and are common for all axes.
Values 16 to 31 are mapped directly to driver inputs that are present on the CN1 connector. They are unique for each axis. It depends on the type of Servo Driver which Servo Driver inputs are mapped into inputs 16 to 31. For more information on Servo Driver I/O mapping into the Trajexia I/O space, refer to section 5-1-4.
Note: The origin input is active low, i.e., the origin switch is set when the input is OFF. The feedhold, reverse jog, forward jog, forward and reverse limit inputs are also active low. Active low inputs are used to enable fail-safe wiring.

Arguments N/A

Example **DATUM_IN AXIS(0)=5**

See also **AXIS**, **DATUM**.

4-2-67 DECEL

Type Axis parameter

Syntax **DECEL**

Description The **DECEL** axis parameter contains the axis deceleration rate. The rate is set in units/s^2. The parameter can have any positive value including 0.

Arguments N/A

Example **DECEL = 100 ' Set deceleration rate**
PRINT " Deceleration rate is ";DECEL;" mm/s/s"

See also **ACCEL**, **AXIS**, **UNITS**.

4-2-68 DEFPOS

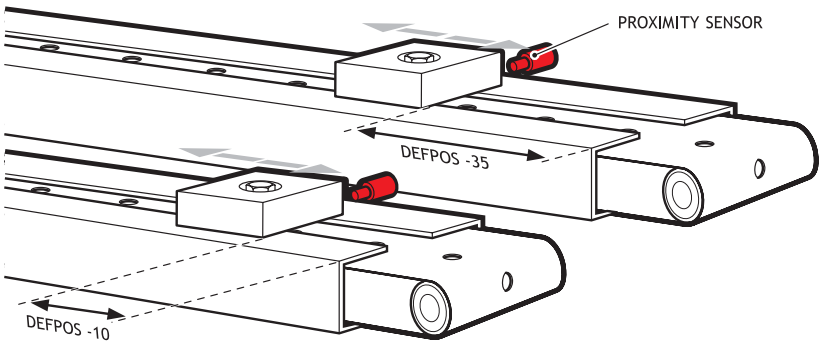
Type Axis command

Syntax **DEFPOS(pos_1 [, pos_2 [, pos_3 [, pos_4 [, ...]]]])**
DP(pos_1 [, pos_2 [, pos_3 [, pos_4 [, ...]]]])

Description The **DEFPOS** command defines the current demand position (**DPOS**) as a new absolute position. The measured position (**MPOS**) will be changed accordingly in order to keep the Following Error. **DEFPOS** is typically used after an origin search sequence (see DATUM command), as this sets the current position to 0. **DEFPOS** can be used at any time. As an alternative also the **OFFPOS** axis parameter can be used. This parameter can be used to perform a relative adjustment of the current position.
DEFPOS works on the default basis axis or axis sequence group (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note: The changes to the axis position made using DEFPOS or OFFPOS are made on the next servo update. This can potentially cause problems when a move is initiated in the same servo period as the DEFPOS or OFFPOS.
The following example shows how the **OFFPOS** parameter can be used to avoid this problem. DEFPOS commands are internally converted into **OFFPOS** position offsets, which provides an easy way to avoid the problem by programming as follows:
DEFPOS(100): WAIT UNTIL OFFPOS = 0: MOVEABS(0)

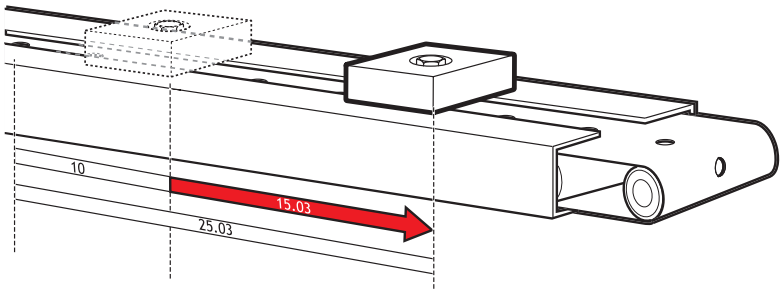
Arguments The command can take up to 16 arguments.

- **pos_i**
  The absolute position for (base+i) axis in user units. Refer to the **BASE** command for the grouping of the axes.



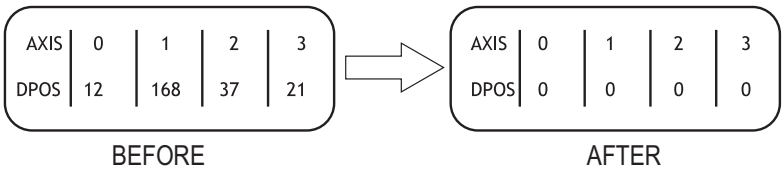
Example After 2 axes returned to their homing positions, it is required to change the **DPOS** values so that the home positions are not zero, but some defined positions instead.

```
DATUM(5) AXIS(1) ' home both axes. At the end of the DATUM
DATUM(4) AXIS(3) ' procedure, the positions are 0,0.
WAIT IDLE AXIS(1)
WAIT IDLE AXIS(3)
BASE(1,3) ' set up the BASE array
DEFPOS(-10,-35) ' define positions of the axes to be -10 and -35
```



Example Set the axis position to 10, then start an absolute move, but make sure the
axis has updated the position before loading the **MOVEABS**.

```
DEFPOS(10.0)
WAIT UNTIL OFFPOS=0
' Makes sure that DEFPOS is complete before next line
MOVEABS(25.03)
```



Example From the Command Line of the Terminal window, quickly set the **DPOS** values of the first four axes to 0.

```
>>BASE(0)
>>DP(0,0,0,0)
```

See also **AXIS**, **DATUM**, **DPOS**, **OFFPOS**, **MPOS**, **UNITS**.

4-2-69 DEL

Type Program command

Syntax **DEL [program_name]**
RM [program_name]

Description The **DEL** command deletes a program from the controller. **DEL** without a program name can be used to delete the currently selected program (using **SELECT**). The program name can also be specified without quotes. **DEL ALL** will delete all programs.
DEL can also be used to delete the Table: **DEL "TABLE"**. The name **"TABLE"** must be in quotes.
Note: This command is implemented for the Command Line Terminal.

Arguments
- **program_name**
  Name of the program to be deleted.

Example **>> DEL oldprog**

See also **COPY**, **NEW**, **RENAME**, **SELECT**, **TABLE**.

4-2-70 DEMAND_EDGES

Type Axis parameter (read-only)

Syntax **DEMAND_EDGES**

Description The **DEMAND_EDGES** axis parameter contains the current value of the **DPOS** axis parameter in encoder edge units.

Arguments N/A

Example No example.

See also **AXIS**, **DPOS**.

4-2-71 DIR

Type Program command

Syntax **DIR**
LS

Description The **DIR** command shows a list of the programs held in the controller, the memory size and the **RUNTYPE**. DIR also shows the available memory size, power up mode and current selected program of the controller.
Note: This command is implemented for the Command Line Terminal only.

Arguments N/A

Example No example.

See also **FREE**, **POWER_UP**, **PROCESS**, **RUNTYPE**, **SELECT**.

4-2-72 DISABLE_GROUP

Type Axis command

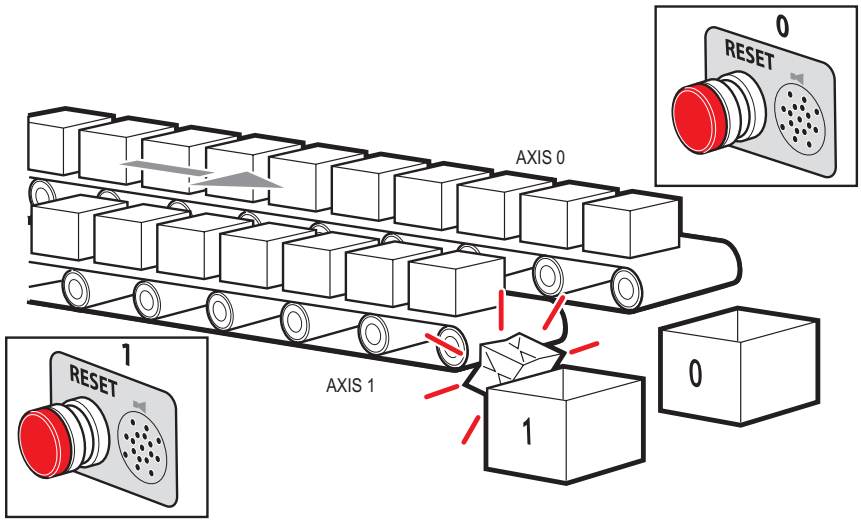
Syntax **DISABLE_GROUP(-1)**
DISABLE_GROUP(axis_1 [, axis_2 [, ...]])

Description The AXIS_ENABLE is used to create a group of axes which will be disabled if there is a motion error in any or more axes in the group. After the group is made, when an error occurs on **one** they will all have their **AXIS_ENABLE** set OFF and **SERVO** set OFF. Multiple groups can be made, although an axis cannot belong to more than one group. All groupings can be cleared using **DISABLE_GROUP(-1)**.
Note: For use with MECHATROLINK-II only.

Arguments
- **axis_i**
  A BASIC expression that evaluates to an axis number.

Example A machine has 2 functionally separate parts, which have their own emergency stop and operator protection guarding. If there is an error on one part of the machine, the other part can continue to run while the cause of the error is removed and the axis group restarted. For this, 2 separate axis groupings must be set up.

```
DISABLE_GROUP(-1) ' remove any previous axis groupings
DISABLE_GROUP(0,1,2,6) ' group axes 0 to 2 and 6
DISABLE_GROUP(3,4,5,7) ' group axes 3 to 5 and 7
WDOG=ON ' turn on the enable relay and the remote drive enable
FOR ax=0 TO 7
  AXIS_ENABLE AXIS(ax)=ON ' enable the 8 axes
  SERVO AXIS(ax)=ON ' start position loop servo for each axis
NEXT ax
```



Example Two conveyors operated by the same Motion Coordinator are required to run independently, to make sure that the second conveyor does not stop if the first conveyor is blocked.

```
DISABLE_GROUP(0) 'put axis 0 in its own group
DISABLE_GROUP(1) 'put axis 1 in another group
GOSUB group_enable0
GOSUB group_enable1
WDOG=ON
FORWARD AXIS(0)
FORWARD AXIS(1)
WHILE TRUE
  IF AXIS_ENABLE AXIS(0)=0 THEN
    PRINT "motion error axis 0"
    reset_0_flag=1
  ENDIF
  IF AXIS_ENABLE AXIS(1)=0 THEN
    PRINT "motion error axis 1"
    reset_1_flag=1
  ENDIF
  IF reset_0_flag=1 AND IN(0)=ON THEN
    GOSUB group_enable0
    FORWARD AXIS(0)
    reset_0_flag=0
  ENDIF
  IF reset_1_flag=1 AND IN(1)=ON THEN
    GOSUB group_enable1
    FORWARD AXIS(1)
    reset_1_flag=0
  ENDIF
WEND
group_enable0:
  BASE(0)
  DATUM(0) ' clear motion error on axis 0
  WA(10)
  AXIS_ENABLE=ON
RETURN
group_enable1:
  BASE(1)
  DATUM(0) ' clear motion error on axis 0
  WA(10)
  AXIS_ENABLE=ON
  SERVO=ON
RETURN
```

Example One group of axes in a machine must be reset if a motion error occurs, without affecting the remaining axes. This must be done manually by clearing the cause of the error, pressing a button to clear the error flags of the controllers and re-enabling the motion.

```
DISABLE_GROUP(-1) 'remove any previous axis groupings
DISABLE_GROUP(0,1,2) 'group axes 0 to 2
GOSUB group_enable 'enable the axes and clear errors
WDOG=ON
SPEED=1000
FORWARD
WHILE IN(2)=ON
 check axis 0, but all axes in the group will disable together
 IF AXIS_ENABLE =0 THEN
  PRINT "Motion error in group 0"
  PRINT "Press input 0 to reset"
  IF IN(0)=0 THEN 'checks if reset button is pressed
   GOSUB group_enable 'clear errors and enable axis
   FORWARD 'restarts the motion
  ENDIF
 ENDIF
WEND
STOP 'stop program running into sub routine
group_enable: 'Clear group errors and enable axes
 DATUM(0) 'clear any motion errors
 WA(10)
 FOR axis_no=0 TO 2
  AXIS_ENABLE AXIS(axis_no)=ON 'enable axes
  SERVO AXIS(axis_no)=ON 'start position loop servo
 NEXT axis_no
 RETURN
```

See also N/A

4-2-73 DPOS

Type Axis parameter (read-only)

Syntax **DPOS**

Description The **DPOS** axis parameter contains the demand position in user units, which is generated by the move commands in servo control. When the controller is in open loop (**SERVO=OFF**), the measured position (**MPOS**) will be copied to the **DPOS** in order to maintain a 0 Following Error.
The range of the demand position is controlled with the **REP_DIST** and **REP_OPTION** axis parameters. The value can be adjusted without doing a move by using the **DEFPOS** command or **OFFPOS** axis parameter. **DPOS** is reset to 0 at start-up or controller reset.

Arguments N/A

Example
```
>> PRINT DPOS AXIS(0)
34.0000
```
The above line will return the demand position in user units.

See also **AXIS**, **DPOS**, **DEFPOS**, **DEMAND_EDGES**, **FE**, **MPOS**, **REP_DIST**, **REP_OPTION**, **OFFPOS**, **UNITS**.

4-2-74 DRIVE_ALARM

Type Axis command

Syntax **DRIVE_ALARM(VR [,alarm_number])**

Description The **DRIVE_ALARM** function reads the current alarm of the Servo Driver that is connected to the Trajexia system via MECHATROLINK-II. Upon successful execution, the command returns -1 and stores the value in the VR memory location specified by the VR parameter. If the command cannot be executed, the value 0 is returned. The command is executed on the driver for the base axis set by **BASE**. The base axis can be changed with the **AXIS** modifier, as with all the other axis commands and parameters.
This command waits for the response from the axis, The execution of the command can be slow and variable in time. If you require a quick response do not use this command.

Arguments
- **VR**
  The alarm value is stored on the VR address on successful execution.
- **alarm_number**
  Optional parameter to set which alarm to read. 0 means the last alarm (default), 1 means the penultimate alarm, etc. **alarm_number** ranges from 0 to 9.

Example
```
IF NOT DRIVE_ALARM(10) AXIS(2)THEN
  PRINT "Failed to readalarm for Servo Driver"
ELSE
  IF VR(10) = 0THEN
    PRINT "ServoDriver healthy"
  ELSE
    PRINT "Servoalarm code: "; VR(10)
  ENDIF
ENDIF
```
This example reads an alarm of the Servo Driver driving axis 2 and present that information to the user.

See also N/A

4-2-75 DRIVE_CLEAR

Type Axis command

Syntax **DRIVE_CLEAR**

Description The **DRIVE_CLEAR** command clears the alarm status of the Servo Driver connected via the MECHATROLINK-II bus. This command is not capable of clearing all the possible alarm states. Some alarms can only be cancelled by turning off the power supply (both the CJ1W-MCH72 and the Servo Driver), and then turning it on again. Also, an alarm will not be cleared if the cause of the alarm is still present. The command is executed on the driver for the base axis set by BASE. The base axis can be changed with the AXIS modifier, as with all the other axis commands and parameters.

Arguments N/A

Example No example.

See also **DRIVE_STATUS**.

⚠**Caution** Be sure that no Parameter Unit or Personal Computer Software is connected to the Servo Driver when executing this command. Otherwise the program task will be paused until the connection of the other device to the Servo Driver is removed.

4-2-76 DRIVE_CONTROL

Type Axis parameter

Syntax **DRIVE_CONTROL**

Description When applied to an axis driven by the Servo Driver connected to the system via the MECHATROLINK-II bus, this parameter selects the data to be monitored by **DRIVE_MONITOR** according to the table below.

Code	Description
2	Following error (this is the real FE when ATYPE=40 is used)
8	Feedback speed (With ATYPE=41 Units=Max Speed/40000000H, with other ATYPE Units= reference units/s)
9	Command speed (units same as in Feedback Speed)
10	Target speed (units same as in Feedback Speed)
11	Torque (Force) reference (With ATYPE=42 Units=Max Torque/40000000H, with other ATYPE Units= % over nominal Torque
14	Monitor selected with Pn813.0 Useful to monitor servo monitors (Unxxx)
15	Monitor selected with Pn813.1 Useful to monitor servo monitors (Unxxx)

When applied to an axis driven by the Servo Driver connected to the system via the Encoder Interface, this parameter sets outputs of the Encoder Interface. Set bit 8 of this parameter to switch on OUT 0 for an axis. Set bit 9 of this parameter to switch on OUT 1 for an axis. Keep in mind that the same outputs are used by the **HW_PSWITCH** command. The command is executed on the driver for the base axis set by BASE. The base axis can be changed with the AXIS modifier, as with all the other axis commands and parameters.

Arguments N/A

Example **DRIVE_CONTROL AXIS(2) = 256**
In this example, OUT 0 is switched on for axis 2, connected using the Encoder Interface.

See also N/A

4-2-77 DRIVE_INPUTS

Type Axis parameter

Syntax **DRIVE_INPUTS**

Description This parameter monitors the status of the inputs of the Servo Driver connected via the MECHATROLINK-II bus. The parameter value is updated each **SERVO_PERIOD** cycle. It is a bit-wise word with the bits as listed in the table below.

Bit number	Servo Driver input signal			Description
	Sigma-II	Sigma-V	Junma	
0	P_OT	P_OT	P_OT	Forward limit switch
1	N_OT	N_OT	N_OT	Reverse limit switch
2	DEC	DEC	/DEC	Zero point return deceleration
3	PA	PA	Not used	Encoder A phase signal
4	PB	PB	Not used	Encoder B phase signal
5	PC	PC	Not used	Encoder C phase signal
6	EXT1	EXT1	/EXT1	First external latch signal
7	EXT2	EXT2	Not used	Second external latch signal
8	EXT3	EXT3	Not used	Third external latch signal
9	BRK	BRK	/BRK	Brake output
10	Reserved	HBB	E-STP	Emergency stop switch
11	Reserved	Reserved	Not used	
12	IO12	IO12	Not used	Not used
13	IO13	IO13	Not used	Not used
14	IO14	IO14	Not used	Not used
15	IO15	IO15	Not used	Not used

The recommended setting is: Pn81E=4321 & Pn511=654x. Refer to section 5-1-4 for more information about mapping Servo Driver inputs and outputs.
The command is executed on the driver for the base axis set by BASE. The base axis can be changed with the AXIS modifier, as with all the other axis commands and parameters.

Arguments N/A

Example No example.

See also N/A

4-2-78 DRIVE_MONITOR

Type Axis parameter

Syntax **DRIVE_MONITOR**

Description This parameter contains the monitored data of the Servo Driver connected to the system via the MECHATROLINK-II bus. The data to be monitored is selected using **DRIVE_CONTROL** and can be displayed in the Trajexia Studio scope or used inside a program. The monitored data is updated each **SERVO_PERIOD**. The command is executed on the driver for the base axis set by BASE. The base axis can be changed with the AXIS modifier, as with all the other axis commands and parameters.

Arguments N/A

Example No example.

See also N/A

4-2-79 DRIVE_READ

Type Axis command

Syntax **DRIVE_READ(parameter, size, VR)**

Description The **DRIVE_READ** function reads the specified parameter of the Servo Driver connected to the Trajexia system via the MECHATROLINK-II bus. Upon successful execution, this command returns -1 and puts the read value in the VR memory location specified by the VR parameter. If the command cannot be executed, the value 0 is returned. The command is executed on the driver for the base axis set with **BASE**. It can be changed using the **AXIS** modifier, like with all the other axis commands and parameters.
Note: This command waits for the response of the axis, therefore its execution is slow and the time variable. Do not use this command together with other commands that require quick execution.
Note: Executing a **DRIVE_READ** will temporarily disable the Servo Driver Front Panel display.
Note: **DRIVE_READ** returns -1 on success. It also returns -1 with no parameter read if the parameter number does not exist or has the wrong size.

Arguments
- **parameter**
  The number of the parameter to be read. Note that the parameter numbers are hexadecimal. The format of the data can be found in the Servo Driver manual.
- **size**
  SIze of the parameter is specified in bytes. For most parameters the size is normally 2 bytes. Some special parameters may be 4 bytes long. Sizes for each parameter can be found in the Servo Driver manual.
- **VR**
  The VR address where the read parameter is stored upon successful execution.

Example
```
IF DRIVE_READ($100,2,1) THEN
  PRINT "The Speed loop gain is: ";VR(1)
ELSE
  PRINT "The speed loop gain could not be read"
ENDIF
```

See also **DRIVE_WRITE**, **HEX**, **$ (HEXADECIMAL INPUT)**.

⚠ **Caution** Be sure that no Parameter Unit or Personal Computer Software is connected to the Servo Driver when executing this command. Otherwise the program task will be paused until the connection of the other device to the Servo Driver is removed.

4-2-80 DRIVE_RESET

Type Axis command

Syntax **DRIVE_RESET**

Description The **DRIVE_RESET** command resets the Servo Driver connected via the MECHATROLINK-II bus. The command is executed on the driver for the base axis set by BASE. The base axis can be changed with the AXIS modifier, as with all the other axis commands and parameters.

Arguments N/A

Example No example.

See also N/A

⚠ **Caution** Be sure that no Parameter Unit or Personal Computer Software is connected to the Servo Driver when executing this command. Otherwise the program task will be paused until the connection of the other device to the Servo Driver is removed.

4-2-81 DRIVE_STATUS

Type Axis parameter (read-only)

Syntax **DRIVE_STATUS**

Description For MECHATROLINK-II axes, this parameter is set from the STATUS field in the MECHATROLINK-II communication frame and is updated every servo period. Those bits can be seen in the drive configuration window in Trajexia Studio, and can be used in programs. The explanation of each bit is given in the table below. (Note: Only bits relevant to MECHATROLINK-II axes are listed.) For the detailed explanation for these status bits, see the MECHATROLINK-II manual.

Bit	Description (MECHATROLINK-II)
0	Alarm
1	Warning
2	Ready
3	Servo on
4	Power on
5	Machine Lock
6	Home Position
7	At Position/Speed
8	Output Completed
9	Torque Limit
10	Latch Completed
11	In Range/Speed Limit

For Flexible Axis axes, this parameter holds the status of registration and auxiliary inputs, as well as registration selection. The explanation of each bit is given in the second table below. (Note: Only bits relevant to Flexible axis are listed.)

Bit	Description (Flexible Axis)
0	MARK
1	MARKB

Bit	Description (Flexible Axis)
2	REG 0 selected current value
3	REG 1 selected current value
5	REG 0 current value
6	REG 1 current value

Arguments N/A

Example **PRINT DRIVE_STATUS AXIS(4)**
This command will print the current value of **DRIVE_STATUS** for axis(4).

Example
```
BASE(3)
ATYPE = 44
IF (DRIVE_STATUS AND 32)= 32 THEN
  PRINT "REG 0 input is ON for axis(3)"
ENDIF
```

See also **AXIS**, **MARK**, **MARKB**, **REGIST**.

4-2-82 DRIVE_WRITE

Type Axis command

Syntax **DRIVE_WRITE(parameter, size, value [,mode])**

Description The **DRIVE_WRITE** function writes to the specified parameter of the Servo Driver via the MECHATROLINK-II bus. Upon successful execution, this command returns -1. If the command cannot be executed, the value 0 is returned. The command is executed on the driver for the base axis set with **BASE**. It can be changed using the **AXIS** modifier, as with all other axis commands and parameters. For some parameters to be written the driver needs to be powered off and on again. The **DRIVE_RESET** command can be used for that purpose.
Note: This command waits for the response of the axis so, its execution is slow and the time variable. Do not use this command together with other commands that require quick execution.
Note: Executing a **DRIVE_WRITE** will temporarily disable the Servo Driver Front Panel display.
Note: **DRIVE_WRITE** returns -1 on success. It also returns -1 with no parameter read if the parameter number does not exist or has the wrong size.

Arguments
- **parameter**
  The number of the parameter to write to. Note that the parameter numbers are hexadecimal. The format of the data can be found in the Refer to the Servo Driver manual for the format of the data.
- **size**
  SIze of the parameter is specified in bytes. For most parameters the size is normally 2 bytes. Some special parameters may be 4 bytes long. Sizes for each parameter can be found in the Servo Driver manual.
- **value**
  The value to be written into driver parameter.
- **mode**
  The write mode. Possible values: 0 (or omitted) - write and store in RAM; 1 - write and store in EPROM.

Example
```
IF DRIVE_WRITE($100,2,90) THEN
  PRINT "The new speed loop gain is: 90"
ELSE
  PRINT "The speed loop gain could not be written in RAM"
ENDIF
```

See also
- **DRIVE_READ**, **DRIVE_RESET**, **$ (HEXADECIMAL INPUT)**

⚠ **Caution** Be sure that no Parameter Unit or Personal Computer Software is connected to the Servo Driver when executing this command. Otherwise the program task will be paused until the connection of the other device to the Servo Driver is removed.

4-2-83 EDIT

Type: Program command

Syntax:
```
EDIT [ line_number ]
ED [ line_number ]
```

Description: The **EDIT** command starts the built in screen editor allowing a program in the controller to be modified using a Command Line Terminal. The currently selected program will be edited.
The editor commands are as follows:
- Quit Editor: [CTRL] K and D
- Delete Line: [CTRL] Y
  This command is implemented for a Command Line Terminal.

Arguments
- **line_number**
  The number of the line at which to start editing.

Example: No example.

See also: **SELECT**.

4-2-84 ELSE

See **IF..THEN..ELSE..ENDIF**.

4-2-85 ELSEIF

See **IF..THEN..ELSE..ENDIF**.

4-2-86 ENCODER

Type Axis parameter (read-only)

Syntax **ENCODER**

Description The **ENCODER** axis parameter contains a raw copy of the encoder hardware register or the raw data received from the drive via MECHATROLINK-II. On axes with absolute encoders, the ENCODER parameter contains a value using a number of bits programmed with ENCODER_BITS.
The **MPOS** axis parameter contains the measured position calculated from the **ENCODER** value automatically, allowing for overflows and offsets.

Arguments N/A

Example No example.

See also **AXIS**, **MPOS**.

4-2-87 ENCODER_BITS

Type Axis parameter

Syntax **ENCODER_BITS = value**

Description This axis parameter configures the interface for the number of encoder bits for Flexible axis SSI and EnDat absolute encoder axes. The parameter is applicable only to axes of **ATYPE** values 47 and 48.
When applied to Flexible axis EnDat absolute encoder axis, bits 0 - 7 of the parameter should be set to the total number of encoder bits. Bits 8 - 14 should be set to the number of multi-turn bits to be used.
When applied to Flexible axis SSI absolute encoder axis, bits 0 - 5 of the parameter should be set to the number of encoder bits. Bit 6 should be 1 for binary operation, or 0 for Gray code.
For SSI encoders of the “Balluff” brand bits 8..10 allow an additional hardware shift to be specified. Normally bits 8..10 are 0.
Note: If using Flexible axis absolute encoder axis, it is essential to set this parameter for the axis before setting the ATYPE.

Arguments N/A

Example **ENCODER_BITS = 25 + (256 * 12)**
ATYPE = 47
In this example a 25 bit EnDat encoder is used, that has 12 bits for multi-turn value and 13 bits per one revolution.

Example **ENCODER_BITS = 12 + (64 * 1)**
ATYPE = 48
In this example a 12 bit (4096 positions per revolution) SSI encoder is used, with binary output type.

See also **AXIS**.

4-2-88 ENCODER_CONTROL

Type Axis parameter

Syntax **ENCODER_CONTROL = value**

Description The **ENCODER_CONTROL** parameter is applicable only to Flexible axis absolute EnDat axis with ATYPE value 47. The parameter controls the mode in which EnDat encoder return its position. The encoder can be set to either cyclically return its position, or it can be set to a parameter read/write mode. The default after initialization is cyclic position return mode. For more information see EnDat absolute encoder interface specification.

Arguments N/A

Example **ENCODER_CONTROL AXIS(1) = 0**
This command sets cyclic position return mode.

Example **ENCODER_CONTROL AXIS(1) = 1**
This command sets parameter read/write mode.

See also **AXIS**, **ENCODER**, **ENCODER_BITS**.

4-2-89 ENCODER_RATIO

Type Axis parameter

Syntax **ENCODER_RATIO(denominator, numerator)**

Description This command allows the incoming encoder count to be scaled by a non integer number, using the equation:
MPOS = (numerator / demoninator) x encoder_edges_input
Unlike the **UNITS** parameters, **ENCODER_RATIO** affects commands like **MOVECIRC** and **CAMBOX**, since it affects the number of encoder edges within the servo loop at the low level. It is necessary to change the position loop gains after changing encoder ratio in order to maintain performance and stability.

Note: Large ratios should be avoided as they will lead to either loss of resolution or much reduced smoothness in the motion. The actual physical encoder count is the basic resolution of the axis and the use of this command may reduce the ability of the Motion Controller to accurately achieve all positions.
Note: ENCODER_RATIO does not replace UNITS. Only use **ENCODER_RATIO** where absolutely necessary. For all other axis scaling use **UNITS**.

Arguments
- **denominator**
  A number between 0 and 16777215 that is used to define the denominator in the above equation.
- **numerator**
  A number between 0 and 16777215 that is used to define the numerator in the above equation.

Example
```
' 7200 is the closest to the encoder resolution that can be devided by an
' integer to give degrees. (7200/20=360)
ENCODER_RATIO(8192,7200)
UNITS=20  ' axis calibrated in degrees, resolution is 0.05 deg.
```
A rotary table has a servo motor connected directly to its centre of rotation. An encoder is mounted to the rear of the servo motor and returns a value of 8192 counts per revolution. The application requires the table to be calibrated in degrees, but so that one degree is an integer number of counts.

See also N/A

4-2-90 ENCODER_READ

Type Axis command

Syntax **ENCODER_READ(address)**

Description The **ENCODER_READ** command is applicable only to Flexible axis absolute EnDat axis with **ATYPE** value 47. The parameter returns a 16-bit encoder parameter stored at specified address. Bits 8 -15 of the address are the EnDat MRS field settings and bits 0 - 7 are the offset within the EnDat MRS block. If a CRC error occurs, this command will return -1. For more information see EnDat absolute encoder interface specification.

Arguments
- **address**
  Specifies the EnDat MRS field to read.

Example **VR(100) = ENCODER_READ($A10D) AXIS(7)**
This command will read the number of encoder bits and put that value in VR(10) memory location.

See also **AXIS**, **ENCODER**, **ENCODER_BITS**.

4-2-91 ENCODER_TURNS

Type Axis parameter (read-only)

Syntax **ENCODER_TURNS**

Description The **ENCODER_TURNS** parameter returns the number of multi-turn count from the encoder.
This is applicable only to Flexible axis absolute EnDat axis with **ATYPE** value 47.
The multi-turn data is not automatically applied to the axis **MPOS** parameter after initialization. The application programmer must apply this from the program using **OFFPOS** or **DEFPOS** commands as required.
If applied to axis of **ATYPE** value other than 47, the parameter returns 0.

Arguments N/A

Example **PRINT ENCODER_TURNS AXIS (1)**
This command will print absolute encoder multi-turn counts for axis 1.

See also **AXIS**, **ENCODER**, **ENCODER_BITS**.

4-2-92 ENCODER_WRITE

Type Axis command

Syntax **ENCODER_WRITE(address, value)**

Description The **ENCODER_WRITE** command is applicable only to Flexible axis absolute EnDat axis with **ATYPE** value 47. The command writes to an encoder parameter specified by the address. Bits 8 -15 of the address are the EnDat MRS field settings and bits 0 - 7 are the offset within the EnDat MRS block. If a CRC error occurs, this command will return 0. Writing to address 0 performs an encoder reset function.For more information see EnDat absolute encoder interface specification.In order to successfully write an encoder parameter with this command, the **ENCODER_CONTROL** parameter must be set to 1, encoder parameter read/write mode.

Arguments
- **address**
  Specifies the EnDat MRS field to write to.
- **value**
  Any valid BASIC expression.

Example No example.

See also **AXIS**, **ENCODER**, **ENCODER_BITS**, **ENCODER_CONTROL**.

4-2-93 ENDIF

See **IF..THEN..ELSE..ENDIF**.

4-2-94 ENDMOVE

Type Axis parameter

Syntax **ENDMOVE**

Description The **ENDMOVE** axis parameter holds the position of the end of the current move in user units. If the **SERVO** axis parameter is ON, the **ENDMOVE** parameter can be written to produce a step change in the demand position (**DPOS**).
Note: As the measured position is not changed initially, the Following Error limit (**FE_LIMIT**) should be considered when writing to ENDMOVE to produce a step change. If the change of demanded position is too big, the limit will be exceeded.

Arguments N/A

Example No example.

See also **AXIS**, **DPOS**, **FE_LIMIT**, **UNITS**.

4-2-95 EPROM

Type Program command

Syntax **EPROM**

Description The **EPROM** command stores the BASIC programs in the CJ1W-MCH72 battery backed up RAM memory in the flash EPROM memory. Whether the programs stored in the flash EPROM memory are copied to RAM at start-up is controlled by the POWER_UP system parameter.
Note: Trajexia Studio offers this command as a command on the **Online** menu.

Arguments N/A

Example No example.

See also **POWER_UP**, **RUNTYPE**.

4-2-96 ERROR_AXIS

Type System parameter (read-only)

Syntax **ERROR_AXIS**

Description The **ERROR_AXIS** axis parameter contains the number of the axis which has caused the motion error.
A motion error occurs when the **AXISSTATUS** state for one of the axes matches the **ERRORMASK** setting. In this case the enable switch (**WDOG**) will be turned off, the **MOTION_ERROR** parameter will have value different than 0 and the **ERROR_AXIS** parameter will contain the number of the first axis to have the error.
Note: The value of **ERROR_AXIS** is not cleared when the error condition is fixed. It is only changed when a new error occurs.

Arguments N/A

Example No example.

See also **AXISSTATUS**, **ERRORMASK**, **MOTION_ERROR**, **WDOG**.

4-2-97 ERROR_LINE

Type Task parameter (read-only)

Syntax **ERROR_LINE**

Description The **ERROR_LINE** parameter contains the number of the line which caused the last BASIC run-time error in the program task. This value is only valid when the **BASICERROR** parameter is **TRUE**.
Each task has its own **ERROR_LINE** parameter. Use the **PROC** modifier to access the parameter for a certain task. Without **PROC**, the current task will be assumed.

Arguments N/A

Example **>> PRINT ERROR_LINE PROC(4)**
23.0000

See also **BASICERROR**, **PROC**, **RUN_ERROR**.

4-2-98 ERRORMASK

Type Axis parameter

Syntax **ERRORMASK**

Description The **ERRORMASK** axis parameter contains a mask value that is **AND**ed bit by bit with the **AXISSTATUS** axis parameter on every servo cycle to determine if a motion error has occurred. If the result of the AND operation is not zero, the motion error has occurred.
When a motion error occurs the enable switch (**WDOG**) will be turned off, the **MOTION_ERROR** parameter will have value different than 0 and the **ERROR_AXIS** parameter will contain the number of the first axis to have the error.
Check the **AXISVALUES** parameter for the status bit allocations. The default setting of **ERRORMASK** is 268.

Arguments	N/A
Example	No example.
See also	**AXIS**, **AXISSTATUS**, **MOTION_ERROR**, **WDOG**.

⚠ **Caution** It is up to the user to define in which cases a motion error is generated. For safe operation it is strongly recommended to generate a motion error when the Following Error has exceeded its limit in all cases. This is done by setting bit 8 of **ERRORMASK**

4-2-99 EX

Type	System command
Syntax	**EX[(option)]**
Description	Resets the controller as if it were being powered up again. There are two types of reset performed by the **EX** command. **EX** without the argument, or **EX(0)** does the software reset of the controller. **EX(1)** does the hardware reset of the controller
Arguments	N/A
Example	No example.
See also	N/A

4-2-100 EXP

Type	Mathematical function
Syntax	**EXP(expression)**
Description	The **EXP** function returns the exponential value of the expression.
Arguments	• **expression** Any valid BASIC expression.
Example	**>>PRINT EXP(1.0)** **2.7183**
See also	N/A

4-2-101 FALSE

Type	Constant (read-only)
Syntax	**FALSE**
Description	The **FALSE** constant returns the numerical value 0.
Arguments	N/A

Example	**test:**
	res = IN(0) OR IN(2)
	IF res = FALSE THEN
	PRINT "Inputs are off"
	ENDIF

See also	N/A

4-2-102 FAST_JOG

Type	Axis parameter
Syntax	**FAST_JOG**
Description	The **FAST_JOG** axis parameter contains the input number to be used as the fast jog input. The number can be from 0 to 31. As default the parameter is set to -1, no input is used for the fast jog. The fast jog input controls the jog speed between two speeds. If the fast jog input is set, the speed as given by the **SPEED** axis parameter will be used for jogging. If the input is not set, the speed given by the **JOGSPEED** axis parameter will be used. Note: This input is active low.
Arguments	N/A
Example	No example.
See also	**AXIS**, **FWD_JOG**, **JOGSPEED**, **REV_JOG**, **SPEED**.

4-2-103 FASTDEC

Type	Axis parameter
Syntax	**FASTDEC**
Description	The **FASTDEC** axis parameter contains fast deceleration ration. Its default value is zero. If a non-zero **FASTDEC** is specified, the axis will ramp to zero at this deceleration rate when an axis limit switch or position is reached.
Arguments	N/A
Example	No example.
See also	N/A

4-2-104 FE

Type	Axis parameter (read-only)
Syntax	**FE**
Description	The **FE** axis parameter contains the position error in user units. This is calculated by the demand position (**DPOS** axis parameter) minus the measured position (**MPOS** axis parameter). The value of the Following Error can be checked by using the axis parameters **FE_LIMIT** and **FE_RANGE**.
Arguments	N/A
Example	No example.
See also	**AXIS**, **DPOS**, **FE_LIMIT**, **FE_RANGE**, **MPOS**, **UNITS**.

4-2-105 FE_LATCH

Type	Axis parameter (read-only)
Syntax	**FE_LATCH**

Description Contains the initial **FE** value which caused the axis to put the controller into MOTION_ERROR. This value is only set when the **FE** exceeds the **FE_LIMIT** and the **SERVO** parameter has been set to OFF. **FE_LATCH** is reset to 0 when the **SERVO** parameter of the axis is set back to ON.

Arguments N/A

Example No example.

See also N/A

4-2-106 FE_LIMIT

Type Axis parameter

Syntax **FE_LIMIT**
FELIMIT

Description The **FE_LIMIT** axis parameter contains the maximum allowed Following Error in user units. When exceeded, bit 8 of the **AXISSTATUS** parameter of the axis will be set. If the **ERRORMASK** parameter has been properly set, a motion error will be generated and WDOG enable relay will be reset to 0.
This limit is used to guard against fault conditions, such as mechanical lock-up, loss of encoder feedback, etc.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **ERRORMASK**, **FE**, **FE_RANGE**, **UNITS**.

4-2-107 FE_LIMIT_MODE

Type Axis parameter

Syntax **FE_LIMIT_MODE=value**

Description When this parameter is set to 0, the axis will cause a **MOTION_ERROR** immediately when the **FE** exceeds the **FE_LIMIT** value.
If **FE_LIMIT_MODE** is set to 1, the axis will only generate a **MOTION_ERROR** when the **FE** exceeds **FE_LIMIT** during 2 consecutive servo periods. This means that if **FE_LIMIT** is exceeded for one servo period only, it will be ignored.
The default value for **FE_LIMIT_MODE** is 0.

Arguments N/A

Example No example.

See also N/A

4-2-108 FE_RANGE

Type Axis parameter

Syntax **FE_RANGE**

Description The **FE_RANGE** axis parameter contains the limit for the Following Error warning range in user units. When the Following Error exceeds this value on a servo axis, bit 1 in the **AXISSTATUS** axis parameter will be turned on.
This range is used as a first indication for fault conditions in the application (compare **FE_LIMIT**).

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **ERRORMASK**, **FE**, **UNITS**.

4-2-109 FHOLD_IN

Type Axis parameter

Syntax **FHOLD_IN**
FH_IN

Description The **FHOLD_IN** axis parameter contains the input number to be used as the feedhold input. The valid input range is 0 to 31. Values 0 to 15 represent physically present inputs of CJ1W-MCH72 I/O connector and are common for all axes.
Values 16 to 31 are mapped directly to driver inputs that are present on the CN1 connector. They are unique for each axis. It depends on the type of Servo Driver which Servo Driver inputs are mapped into inputs 16 to 31. For more information on Servo Driver I/O mapping into the Trajexia I/O space, refer to section 5-1-4.
As default the parameter is set to -1, no input is used for feedhold.
Note: This input is active low.
If an input number is set and the feedhold input turns set, the speed of the move on the axis is changed to the value set in the **FHSPEED** axis parameter. The current move is not cancelled. Furthermore, bit 7 of the **AXISSTATUS** parameter is set. When the input turns reset again, any move in progress when the input was set will return to the programmed speed.
Note: This feature only works on speed controlled moves. Moves which are not speed controlled (**CAMBOX**, **CONNECT** and **MOVELINK**) are not affected.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **FHSPEED**, **UNITS**.

4-2-110 FHSPEED

Type Axis parameter

Syntax **FHSPEED**

Description The **FHSPEED** axis parameter contains the feedhold speed. This parameter can be set to a value in user units/s and defines at which speed the axis will move when the feedhold input turns on. The current move is not cancelled. **FHSPEED** can have any positive value including 0. The default value is 0. This default value is applicable to most applications as motion is usually ramped down to zero speed when the freehold input is set. In some cases it may be desirable for the axis to ramp to a known constant speed when the freehold input is set.
Note: This feature only works on speed controlled moves. Moves which are not speed controlled (**CAMBOX**, **CONNECT** and **MOVELINK**) are not affected.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **FHOLD_IN**, **UNITS**.

4-2-111 FINS_COMMS

Type Communication command

Syntax **FINS_COMMS(type, network, node, unit, remote_area, remote_offset, length, local_area, local_offset, timeout)**

Description FINS (Factory Interface Network Service) is a Proprietary OMRON communication protocol. A subset of this protocol has been implemented in Trajexia. The FINS protocol has been implemented with the intention of enabling seamless communication with other OMRON devices (PLCs, HMIs, etc.) and software (CX-Drive, CX-Server, etc.). For more information on FINS communication protocol, see the Communication Commands Reference Manual, cat. num. W342-E1, Sections 3 and 5.
Trajexia has built in FINS client capabilities, so it can initiate the FINS communications with FINS slave devices using **FINS_COMMS**. Only FINS 0101 (Read Memory) and FINS 0102 (Write Memory) commands are implemented. With FINS 0101, memory can be read from other devices with FINS server capability. FINS 0102 can be used to write data to devices with FINS server capability.
This command returns one of the following values, depending on outcome of the execution:
-1: The command executed successfully.
0: The command failed.
1: Request not sent because the client or the FINS protocol is busy.
2: One or more of the request parameters are invalid.
3: Invalid source memory area.
4: Request was sent, but no response from remote server received within timeout period.
5: Error response code received from remote server.

Arguments

- **type**
  The type of the FINS command. 0 means FINS 0101, read memory from remote FINS server. 1 means FINS 0102, write memory to the remote server.
- **network**
  The destination network. For more details, see the Communication Commands Reference Manual, cat. num. W342-E1, Section 3.
- **node**
  The node of the destination FINS server. For more details, see the Communication Commands Reference Manual, cat. num. W342-E1, Section 3.
- **unit**
  The unit number of the destination FINS server. For more details, see the Communication Commands Reference Manual, cat. num. W342-E1, Section 3.
- **remote_area**
  The area of memory accessed on the destination FINS server. Range: 128..255. Note that this area must be one of the following values if the destination is another Trajexia system: 0xB0: Integer VR value; 0x82: Integer TABLE value; 0xC2: float TABLE value.
- **remote_offset**
  The memory offset on the destination FINS server. Range: 0..65535. Note that this range will be more limited to the maximum TABLE or VR addresses if the destination is another Trajexia system.
- **length**
  The number of items to be transferred. The range will depend upon the FINS frame length and the capabilities of the client and remote servers. The range for a Trajexia system is from 1 to 700 integer values, or 1 to 350 floating point values.
- **local_area**
  The local (source) memory area. Note that this area must be one of the following values: 0x00: Integer VR value; 0x01: Integer TABLE value; 0x02 : float TABLE value.
- **local_offset**
  The offset of the first value in the local (source) memory area. The range depends upon the VR or TABLE array size and value for the length argument.
- **timeout**
  The number of milliseconds to wait for a response from the destination FINS server, before timing out.

Example

FINS_COMMS(0, 0, 0, 0, $82, 1000, 20, 0, 500, 5000)
This command reads 20 words (**length**=20) of DM PLC memory area (**remote_area**=$82), starting from DM1000 (**remote_offset**=1000), and writes it in the CJ1W-MCH72 VR memory in integer format (**local_area**=0), starting from VR(500) (**local_offset**=500). Thus, values in PLC memory range DM1000 to DM1019 are placed in CJ1W-MCH72 memory VR(500) to VR(519). The timeout is set to 5 seconds.

Example

FINS_COMMS(1, 0, 0, 0, $80, 50, 10, 0, 300, 3000)
This command writes 10 words (**length**=10) of CJ1W-MCH72 VR memory as integers (**local_area**=0), starting from VR(300) (**local_offset**=300) to the CIO area of the PLC (**remote_area**=$80), starting from CIO50 (**remote_offset**=50). Thus, the values in the CJ1W-MCH72 memory range VR(300) to VR(309) are placed in memory CIO50 to CIO59 of the PLC. The timeout is set to 3 seconds.

See also N/A

4-2-112 FLAG

Type System command

Syntax **FLAG(flag_number [,value])**

Description The **FLAG** command is used to set and read a bank of 24 flag bits. The **FLAG** command can be used with one or two parameters. With one parameter specified the status of the given flag bit is returned. With two parameters specified the given flag is set to the value of the second parameter. The **FLAG** command is provided to aid compatibility with earlier controllers and is not recommended for new programs.

Arguments
- **flag_number**
  The flag number is a value from 0..23.
- **value**
  If specified this is the state to set the given flag to i.e. ON or OFF. This can also be written as 1 or 0.

Example **FLAG(21,ON)**
Set flag bit 21 on.

See also N/A

4-2-113 FLAGS

Type System command

Syntax **FLAGS([value])**

Description Read and set the **FLAGS** as a block. The **FLAGS** command is provided to aid compatibility with earlier controllers and is not recommended for new programs. The 24 flag bits can be read with **FLAGS** and set with **FLAGS(value)**.

Arguments
- **value**
  The decimal equivalent of the bit pattern to which the flags must be set. See the table below.

Bit number	Decimal value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Example **FLAGS(146) ' 2 + 16 + 128**
Set Flags 1,4 and 7 on, all others off.

Example **IF (FLAGS and 8) <>0 then GOSUB somewhere**
Test if Flag 3 is set.

See also N/A

4-2-114 FOR..TO..STEP..NEXT

Type Program control command

Syntax
```
FOR variable = start TO end [STEP increment]
  commands
NEXT variable
```

Description The **FOR ... NEXT** loop allows the program segment between the **FOR** and the **NEXT** statement to be repeated a number of times.
On entering this loop, the variable is initialized to the value of start and the block of commands is then executed. Upon reaching the **NEXT** command, the variable is increased by the increment specified after **STEP**. The **STEP** value can be positive or negative, if omitted the value is assumed to be 1.
While variable is less than or equal to end, the block of commands is repeatedly executed until variable is greater than end, at which time program execution will continue after **NEXT**.
Note: **FOR ... NEXT** statements can be nested up to 8 levels deep in a BASIC program.

Arguments
- **variable**
  Any valid BASIC expression.
- **start**
  Any valid BASIC expression.
- **end**
  Any valid BASIC expression.
- **increment**
  Any valid BASIC expression.
- **commands**
  One or more BASIC commands.

Example
```
FOR opnum = 8 TO 13
  OP(opnum,ON)
NEXT opnum
```
This loop turns on outputs 8 to 13.

Example
```
loop:
  FOR dist = 5 TO -5 STEP -0.25
    MOVEABS(dist)
    GOSUB pick_up
  NEXT dist
```
The **STEP** increment can be positive or negative.

Example
```
loop1:
  FOR I1 = 1 TO 8
  loop2:
    FOR I2 = 1 TO 6
      MOVEABS(I1*100,I2*100)
      GOSUB 1000
    NEXT I2
  NEXT I1
```
FOR..TO..STEP..NEXT statements can be nested (up to 8 levels deep) provided the inner **FOR** and **NEXT** commands are both within the outer **FOR..TO..STEP..NEXT** loop.

See also **REPEAT..UNTIL**, **WHILE..WEND**.

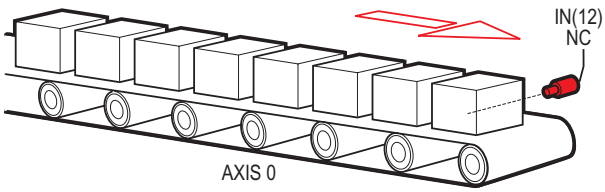
4-2-115 FORWARD

Type Axis command

Syntax **FORWARD**
FO

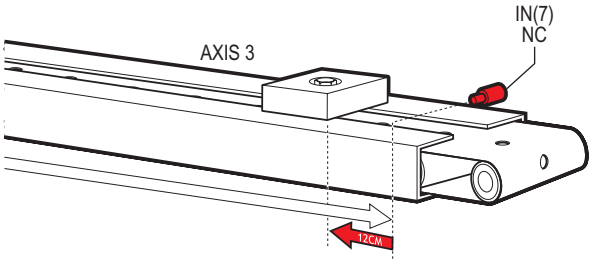
Description The **FORWARD** command moves an axis continuously forward at the speed set in the **SPEED** axis parameter. The acceleration rate is defined by the **ACCEL** axis parameter.
FORWARD works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note: The forward motion can be stopped by executing the **CANCEL** or **RAPIDSTOP** command, or by reaching the forward limit. If stopped by execution of the **CANCEL** or **RAPIDSTOP** command, the axis decelerates to a stop at the programmed DECEL rate.

Arguments N/A



Example Run an axis forwards. When an input signal is detected on input 12, bring the axis to a stop.

```
FORWARD
' wait for stop signal
WAIT UNTIL IN(12)=ON
CANCEL
WAIT IDLE
```



Example Move an axis forward until it hits the end limit switch, then move it in the reverse direction for 25 cm.

```
BASE(3)
FWD_IN=7  limit switch connected to input 7
FORWARD
WAIT IDLE ' wait for motion to stop on the switch
MOVE(-25.0)
WAIT IDLE
```

Example A machine that applies lids to cartons uses a simulated line shaft. This example sets up a virtual axis running forward to simulate the line shaft. Axis 0 is then connected, with the **CONNECT** command, to this virtual axis to run the conveyor. Axis 1 controls a vacuum roller that feeds the lids on to the cartons using the **MOVELINK** control.

```
BASE(4)
ATYPE=0 'Set axis 4 to virtual axis
REP_OPTION=1
SERVO=ON
FORWARD 'starts line shaft
BASE(0)
CONNECT(-1,4) 'Connects base 0 to virtual axis in reverse
WHILE IN(2)=ON
  BASE(1)
  'Links axis 1 to the shaft in reverse direction
  MOVELINK(-4000,2000,0,0,4,2,1000)
  WAIT IDLE
WEND
RAPIDSTOP
```

See also **AXIS**, **CANCEL**, **RAPIDSTOP**, **REVERSE**, **UNITS**.

4-2-116 FPGA_VERSION

Type	Slot parameter
Syntax	**FPGA_VERSION**
Description	This parameter returns the FPGA version of the CJ1W-MCH72.
Arguments	N/A
Example	N/A
See also	N/A

4-2-117 FRAC

Type	Mathematical function
Syntax	**FRAC(expression)**
Description	The **FRAC** function returns the fractional part of the expression.
Arguments	• **expression** Any valid BASIC expression.
Example	**>> PRINT FRAC(1.234)** **0.2340**
See also	N/A

4-2-118 FRAME

Type	System parameter
Syntax	**FRAME=value**
Description	Used to specify which frame to operate within when employing frame transformations. Frame transformations are used to allow movements to be specified in a multi-axis coordinate frame of reference which do not correspond one-to-one with the axes. An example is a SCARA robot arm with jointed axes. For the end tip of the robot arm to perform straight line movements in X-Y the motors need to move in a pattern determined by the robots geometry. Frame transformations to perform functions such as these need to be compiled from C language source and loaded into the controller system software. Contact OMRON if you need to do this. A machine system can be specified with several different frames. The currently active “frame” is specified with the **FRAME** System parameter. The default **FRAME** is 0 which corresponds to a one-to-one transformation.
Arguments	N/A
Example	**FRAME=1**
See also	N/A

4-2-119 FREE

Type	System function
Syntax	**FREE**

Description The **FREE** function returns the remaining amount of memory available for user programs and TABLE array elements.
Note: Each line takes a minimum of 4 characters (bytes) in memory. This is for the length of this line, the length of the previous line, number of spaces at the beginning of the line and a single command token. Additional commands need one byte per token; most other data is held as ASCII.
The CJ1W-MCH72 compiles programs before they are executed, this means that a little under twice the memory is required to be able to execute a program.

Arguments N/A

Example
```
>> PRINT FREE
47104.0000
```

See also DIR, TABLE

4-2-120 FS_LIMIT

Type Axis parameter

Syntax **FS_LIMIT**
FSLIMIT

Description The **FS_LIMIT** axis parameter contains the absolute position of the forward software limit in user units.
A software limit for forward movement can be set from the program to control the working range of the machine. When the limit is reached, the CJ1W-MCH72 will ramp down the speed of an axis to 0, and then cancel the move. Bit 9 of the **AXISSTATUS** axis parameter will be turned on while the axis position is greater than **FS_LIMIT**.
FS_LIMIT is disabled when it has a value greater than **REP_DIST**.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, REP_DIST, **UNITS**.

4-2-121 FWD_IN

Type Axis parameter

Syntax **FWD_IN**

Description The **FWD_IN** axis parameter contains the input number to be used as a forward limit input. The valid input range is 0 to 31. Values 0 to 15 represent physically present inputs of CJ1W-MCH72 I/O connector and are common for all axes.
Values 16 to 31 are mapped directly to driver inputs that are present on the CN1 connector. They are unique for each axis. It depends on the type of Servo Driver which Servo Driver inputs are mapped into inputs 16 to 31. For more information on Servo Driver I/O mapping into the Trajexia I/O space, refer to section 5-1-4.
For more information on setting driver parameter Pn81E, see Servo Driver manual. As default the parameter is set to -1, no inputs selected.
If an input number is set and the limit is reached, any forward motion on that axis will be stopped. Bit 4 of the **AXISSTATUS** will also be set.
Note: This input is active low.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **REV_IN**.

4-2-122 FWD_JOG

Type Axis parameter

Syntax **FWD_JOG**

Description The **FWD_JOG** axis parameter contains the input number to be used as a jog forward input. The input can be set from 0 to 31. As default the parameter is set to -1, no input is selected.
Note: This input is active low.

Arguments N/A

Example No example.

See also **AXIS**, **FAST_JOG**, **JOGSPEED**, **REV_JOG**.

4-2-123 GET

Type I/O command

Syntax **GET [#n,] variable**

Description The **GET** command assigns the ASCII code of a received character to a variable. If the serial port buffer is empty, program execution will be paused until a character has been received. Channels 5 to 7 are logical channels that are superimposed on the programming port 0 when using Trajexia Studio.
Note: Channel 0 is reserved for the connection to Trajexia Studio and/or the command line interface. Please be aware that this channel may give problems for this function.

Arguments
- **n**
  The specified input device. When this argument is omitted, the port is 0 (Terminal window). See the table below.

Input device number	Description
0	Programming port 0
5	Trajexia Studio port 0 user channel 5
6	Trajexia Studio port 0 user channel 6
7	Trajexia Studio port 0 user channel 7

- **variable**
  The name of the variable to receive the ASCII code.

Example **GET#5, k**
This line stores the ASCII character received on the Trajexia Studio port channel 5 in **k**.

See also N/A

4-2-124 GLOBAL

Type: System command

Syntax: **GLOBAL "name", vr_number**

Description: Declares the name as a reference to one of the global VR variables. The name can then be used both within the program containing the **GLOBAL** definition and all other programs in the Trajexia Studio solution.
Note: The program containing the **GLOBAL** definition must be run before the name is used in other programs. In addition, only that program should be running at the time the GLOBAL is executed, otherwise the program error will appear and the program will stop when trying to execute this command. For fast startup the program should also be the only process running at power-up.
When the GLOBAL is declared, the declaration remains active until the next CJ1W-MCH72 reset by switching the power off and back on, or by executing the EX command.
In programs that use the defined **GLOBAL**, **name** has the same meaning as **VR(vr_number)**. Do not use the syntax: **VR(name)**.
A maximum of 128 **GLOBAL**s can be declared.

Arguments:
- **name**
  Any user-defined name containing lower case alpha, numerical or underscore characters.
- **vr_number**
  The number of the VR to be associated with **name**.

Example:
```
GLOBAL "srew_pitch",12
GLOBAL "ratio1",534
ratio1 = 3.56
screw_pitch = 23.0
PRINT screw_pitch, ratio1
```

See also: N/A

4-2-125 GOSUB..RETURN

Type: Program control command

Syntax:
```
GOSUB label
...
RETURN
```

Description: The **GOSUB** structure enables a subroutine jump. **GOSUB** stores the position of the line after the **GOSUB** command and then jumps to the specified label. Upon reaching the **RETURN** statement, program execution is returned to the stored position.
Note: Subroutines on each task can be nested up to 8 levels deep.

Arguments:
- **label**
  A valid label that occurs in the program. An invalid label will give a compilation error before execution.
  Labels can be character strings of any length, but only the first 15 characters are significant. Alternatively line numbers may be used as labels.

Example
```
main:
  GOSUB routine
  GOTO main
routine:
  PRINT "Measured position=";MPOS;CHR(13);
  RETURN
```

See also **GOTO**

4-2-126 GOTO

Type Program control command

Syntax **GOTO label**

Description The **GOTO** structure enables a jump of program execution. **GOTO** jumps program execution to the line of the program containing the label.

Arguments
- **label**
  A valid label that occurs in the program. An invalid label will give a compilation error before execution.
  Labels can be character strings of any length, but only the first 15 characters are significant. Alternatively line numbers may be used as labels.

Example
```
loop:
  PRINT "Measured position = ";MPOS;CHR(13);
  WA(1000)
GOTO loop
```

See also **GOSUB..RETURN**

4-2-127 HALT

Type System command

Syntax **HALT**

Description The **HALT** command stops execution of all program tasks currently running. The command can be used both on Command Line Terminal as in programs. The **STOP** command can be used to stop a single program task.
Note: **HALT** doesn't stop any motion. Currently executing, or buffered moves will continue unless they are terminated with a CANCEL or RAPIDSTOP command.

Arguments N/A

Example No example.

See also **PROCESS**, **STOP**.

4-2-128 HEX

Type I/O command

Syntax **HEX**

Description This command is used in a print statement to output a number in hexadecimal format.

Arguments N/A

Example **PRINT#5,HEX(IN(8,16))**

See also N/A

4-2-129 HW_PSWITCH

Type Axis command

Syntax **HW_PSWITCH(mode, direction, opstate, table_start, table_end)**

Description The **HW_PSWITCH** command turns on digital output 8 for the axis when the predefined axis measured position is reached, and turns the output off when another measured position is reached. Positions are defined as sequence in the TABLE memory in range from **table_start**, to **table_end**, and on execution of the **HW_PSWITCH** command are stored in FIFO queue.
This command is applicable only to Flexible axis axes with **ATYPE** values 43, 44 and 45.
The command can be used with either 1 or 5 parameters. Only 1 parameter is needed to disable the switch or clear FIFO queue. All five parameters are needed to enable switch.
After loading FIFO and going through the sequence of positions in it, if the same sequence has to be executed again, FIFO must be cleared before executing **HW_PSWITCH** command with the same parameters.

Arguments
- **mode**
  0 = disable switch; 1 = on and load FIFO; 2 = clear FIFO.
- **direction**
  0 = decreasing; 1 = increasing.
- **opstate**
  Output state to set in the first position in the FIFO; ON or OFF.
- **table_start**
  Starting TABLE address of the sequence.
- **table_end**
  Ending TABLE address of the sequence.

Example **HW_PSWITCH(1, 1, ON, 21, 50)**
This command will load FIFO with 30 positions, stored in TABLE memory starting from **TABLE(21)** in increasing direction. When the position stored in **TABLE(21)** is reached, output 8 will be set ON and then alternatively OFF and ON on reaching following positions in the sequence, until the position stored in **TABLE(50)** reached.

Example **HW_PSWITCH(0)**
This command will disable switch if it was enabled previously, but will not clear the FIFO queue.

Example **HW_PSWITCH(2)**
This command will clear FIFO queue if loaded previously.

See also **AXIS**

4-2-130 I_GAIN

Type Axis parameter

Syntax **I_GAIN**

Description The **I_GAIN** parameter contains the integral gain for the axis. The integral output contribution is calculated by multiplying the sums of the Following Errors with the value of the **I_GAIN** parameter. The default value is 0.
Adding integral gain to a servo system reduces positioning error when at rest or moving steadily, but it can produce or increase overshooting and oscillation and is therefore only suitable for systems working on constant speed and with slow accelerations.
Note: In order to avoid any instability the servo gains should be changed only when the **SERVO** is off.
Note: Servo gains have no affect on stepper output axis, ATYPE=46.

Arguments N/A

Example No example.

See also **D_GAIN**, **OV_GAIN**, **P_GAIN**, **VFF_GAIN**.

4-2-131 IDLE

See **WAIT IDLE**.

4-2-132 IEEE_IN

Type Mathematical function

Syntax **IEEE_IN(byte0,byte1,byte2,byte3)**

Description The **IEEE_IN** function returns the floating point number represented by 4 bytes which typically have been received over a communications link, such as ModbusTCP or FINS.
Note: byte0 is the high byte of the 32 bit IEEE floating point format.

Arguments
- **byte0** - byte3
  Any combination of 8 bit values that represents a valid IEEE floating point number.

Example **VR(20) = IEEE_IN(b0,b1,b2,b3)**

See also N/A

4-2-133 IEEE_OUT

Type Mathematical function

Syntax **byte_n = IEEE_OUT(value, n)**

Description The **IEEE_OUT** function returns a single byte in IEEE format extracted from the floating point value for transmission over a communications link. The function will typically be called 4 times to extract each byte in turn.
Note: Byte 0 is the high byte of the 32 bit IEEE floating point format.

Arguments
- **value**
  Any BASIC floating point variable or parameter.
- **n**
  The byte number (0 - 3) to be extracted.

Example
```
a=MPOS AXIS(2)
byte0 = IEEE_OUT(a, 0)
byte1 = IEEE_OUT(a, 1)
byte2 = IEEE_OUT(a, 2)
byte3 = IEEE_OUT(a, 3)
```

See also N/A

4-2-134 IF..THEN..ELSE..ENDIF

Type Program control command

Syntax **IF condition_1 THEN commands {ELSEIF condition_i THEN commands} [ELSE commands] ENDIF**
IF condition_1 THEN commands

Description This structure controls the flow of the program based on the results of the condition. If the condition is **TRUE** the commands following **THEN** up to **ELSEIF**, **ELSE** or **ENDIF** are executed. If the condition is **FALSE** and the command of a subsequent **ELSEIF** substructure is **TRUE**, the commands of this substructure are executed. If all conditions are **FALSE** the commands following **ELSE** will be executed or the program will resume at the line after **ENDIF** in case no **ELSE** is included. The **ENDIF** is used to mark the end of the conditional block.
Note: **IF..THEN..ELSE..ENDIF** sequences can be nested without limit. For a multi-line **IF..THEN** construction, there must not be any statement after **THEN**. A single-line construction must not use **ENDIF**.

Arguments
- **condition_i**
  A logical expression.
- **commands**
  One or more BASIC commands.

Example
```
IF MPOS > (0.22 * VR(0)) THEN GOTO exceeds_length
```

Example
```
IF IN(0) = ON THEN
 count = count + 1
 PRINT "COUNTS = ";count
 fail = 0
ELSE
 fail = fail + 1
ENDIF
```

Example
```
IF IN(stop)=ON THEN
 OP(8,ON)
 VR(cycle_flag)=0
ELSEIF IN(start_cycle)=ON THEN
 VR(cycle_flag)=1
ELSEIF IN(step1)=ON THEN
 VR(cycle_flag)=99
ENDIF
```

Example
```
IF key_char=$31 THEN
  GOSUB char_1
ELSEIF key_char=$32 THEN
  GOSUB char_2
ELSEIF key_char=$33 THEN
  GOSUB char_3
ELSE
  PRINT "Character unknown"
ENDIF
```

See also N/A

4-2-135 IN

Type I/O function

Syntax **IN(input_number [,final_input_number])**
IN

Description The **IN** function returns the value of digital inputs.

- **IN(input_number, final_input_number)** will return the binary sum of the group of inputs in range [**input_number**, **final_input_number**]. The two arguments must be less than 24 apart.
- **IN(input_number)** will return the value of the particular input specified by the parameter **input_number**.

Arguments

- **input_number**
  The number of the input for which to return a value. The range for this parameter depends on the number of additional digital I/O connected to the PLC. If there are no digital I/O connected, the range for this parameter is 0..31.
- **final_ input_number**
  The number of the last input for which to return a value. The range for this parameter depends on the number of additional digital I/O connected to the PLC. If there are no digital I/O connected, the range for this parameter is 0..31

Example The following lines can be used to move to the position set on a thumb wheel multiplied by a factor. The thumb wheel is connected to inputs 4, 5, 6 and 7, and gives output in BCD.

```
moveloop:
  MOVEABS(IN(4,7)*1.5467)
  WAIT IDLE
  GOTO moveloop
```

The **MOVEABS** command is constructed as follows:
Step 1: **IN(4,7)** will get a number between 0 and 15.
Step 2: The number is multiplied by 1.5467 to get required distance.
Step 3: An absolute move is made to this position.

Example In this example a single input is tested:

```
test:
  WAIT UNTIL IN(4)=ON ' Conveyor is in position when ON
  GOSUB place
```

See also **OP**.

4-2-136 INITIALISE

Type	System command
Syntax	**INITIALISE**
Description	Sets all axes, system and process parameters to their default values. The parameters are also reset each time the controller is powered up, or when an **EX** (software reset) command is performed. In Trajexia Studio the menu **Reset Device** on the **Online** menu performs the equivalent of an **EX** command.
Arguments	N/A
Example	No example.
See also	• **EX**

4-2-137 INT

Type	Mathematical function
Syntax	**INT(expression)**
Description	The **INT** function returns the integer part of the expression. Note: To round a positive number to the nearest integer value take the **INT** function of the value added by 0.5. Similarly, to round for a negative value subtract 0.5 to the value before applying **INT**.
Arguments	• **expression** Any valid BASIC expression.
Example	**>> PRINT INT(1.79)** **1.0000**
See also	N/A

4-2-138 INVERT_IN

Type	System command
Syntax	**INVERT_IN(input, on/off)**
Description	The **INVERT_IN** command allows the input channels 0..31 to be individually inverted in software. This is important as these input channels can be assigned to activate functions such as feedhold. The **INVERT_IN** function sets the inversion for one channel ON or OFF. It can only be applied to inputs 0..31.
Arguments	• **input** Any valid BASIC expression
Example	**>>? IN(3)** **0.0000** **>>INVERT_IN(3,ON)** **>>? IN(3)** **1.0000**
See also	N/A

4-2-139 INVERT_STEP

Type Axis parameter

Syntax **INVERT_STEP**

Description **INVERT_STEP** is used to switch a hardware Inverter into the stepper pulse output circuit. This can be necessary for connecting to some stepper drivers. The electronic logic inside the Trajexia stepper pulse generator assumes that the FALLING edge of the step output is the active edge which results in motor movement. This is suitable for the majority of stepper drivers. Setting **INVERT_STEP=ON** effectively makes the RISING edge of the step signal the active edge. **INVERT_STEP** should be set if required prior to enabling the controller with **WDOG=ON**. Default is off.
Note: If the setting is incorrect a stepper motor may lose position by one step when changing direction.
Note: This parameter is applicable only to Flexible axis stepper output axes with ATYPE=46. With other types of axes, this parameter has no effect.

Arguments N/A

Example No example.

See also N/A

4-2-140 INVERTER_COMMAND

Type System command

Syntax **INVERTER_COMMAND(0, station, 1, alarm_number)**
INVERTER_COMMAND(0, station, 8, mode)
INVERTER_COMMAND(0, station, 7, operation_signals)

Description **INVERTER_COMMAND** controls inputs and clears alarm of the Inverter connected to the system via the MECHATROLINK-II bus.
There are three **INVERTER_COMMAND** functions:

- 1: Clears an alarm.
- 7: Controls operation signals.
- 8: Set an Inverter to Servo Driver mode, so it acts as a servo axis. This is possible only for Inverters with an encoder feedback option card connected.

To use an Inverter via MECHATROLINK-II you must put the command and the reference via communication option:

- Inverter MV/V7: N3=3; N4=9
- Inverter F7/G7: B1-01=3; B1-02=3.

Make sure that the Inverter firmware supports the MECHATROLINK-II board.
The command returns -1 if successfully executed and 0 if failed.
The command sent to the Inverter corresponds with the bits given in the table below.

Bit	Value	Command	Description
0	Hex	1	Run forward
1	Hex	2	Run reverse

Bit	Value	Command	Description
2	Hex	4	Inverter multifunction Input 3
3	Hex	8	Inverter multifunction Input 4
4	Hex	10	Inverter multifunction Input 5
5	Hex	20	Inverter multifunction Input 6
6	Hex	40	Inverter multifunction Input 7
7	Hex	80	Inverter multifunction Input 8 (Only G7)
8	Hex	100	External fault
9	Hex	200	Fault reset
10	Hex	400	Inverter multifunction Input 9 (only G7)
11	Hex	800	Inverter multifunction Input 10 (only G7)
12	Hex	1000	Inverter multifunction Input 11 (only G7)
13	Hex	2000	Inverter multifunction Input 12 (only G7)
14	Hex	4000	Fault history data clear
15	Hex	8000	External BB command

If with function 8 the mode parameter is set to 1, the Inverter is set into servo axis mode. The corresponding axis number is assigned by the CJ1W-MCH72 using the formula:

AxisNo = MECHATROLINK-II Station Number - 0x21

Therefore the calculated AxisNo must not be occupied by another axis connected.

If with function 8 the mode parameter is set to 0, which is the default value at power-up, the Inverter is set into normal Inverter mode.

Arguments

- **station**
  The MECHATROLINK-II station number of the Inverter.
- **alarm_number**
  The number of the alarm. See the Inverter manual.
- **operation_signals**
  A bitwise value to control the operation signals. See the table below.
- **mode**
  The mode to set the Inverter to:
  0 = Inverter mode. This is the default value at power-up.
  1 = Servo Driver mode.

Example

```
>>INVERTER_WRITE(0,$23,2,4500)
>>INVERTER_COMMAND(0,$23,7,2)
>>WA(10000)
>>INVERTER_COMMAND(0,$23,7,0)
```

The sequence above controls an Inverter connected via MECHATROLINK-II bus to station number 23 (hex), using following steps:
Step 1: Speed reference is set to 45.00 Hz.
Step 2: The Inverter is set to run in reverse direction for 10 seconds with speed reference defined in previous step.
Step 3: The Inverter is stopped.

See also N/A

4-2-141 INVERTER_READ

Type System command

Syntax **INVERTER_READ(0, station, 0, param_number, param_size, VR)**
INVERTER_READ(0, station, 1, alarm_number, VR)
INVERTER_READ(0, station, 2, VR)
INVERTER_READ(0, station, 3, VR)
INVERTER_READ(0, station, 4, from, length, VR)

Description **INVERTER_READ** reads the parameter, speed reference, torque reference or alarm from the Inverter connected to the system via the MECHATROLINK-II bus.
There are five **INVERTER_READ** functions:

- 0: Reads an Inverter parameter.
- 1: Reads the Inverter alarm.
- 2: Reads the speed reference.
- 3: Reads the torque reference.
- 4: Reads the Inverter inputs.

To use an Inverter via MECHATROLINK-II you must put the command and the reference via communication option:

- Inverter MV/V7: N3=3; N4=9
- Inverter F7/G7: B1-01=3; B1-02=3.

Make you sure that the Inverter firmware supports the MECHATROLINK-II board.
The command returns 1 if successfully executed and 0 if failed. The result (if any) is returned in the selected VR.

Arguments

- **station**
  The MECHATROLINK-II station number of the Inverter.
- **param_number**
  The number of the parameter to read. See the Inverter manual.
- **param_size**
  The size of the parameter to read, 2 or 4 bytes. Most of the Inverter parameters are 2 bytes long. See the Inverter manual.
- **VR**
  The address in the VR memory of the CJ1W-MCH72 where the read information is put. When the function is **4**, the result is returned as a bitwise value. See the table below.

Bit	Value	Command	Description
0	Hex	1	Run forward
1	Hex	2	Run reverse
2	Hex	4	Inverter multifunction Input 3
3	Hex	8	Inverter multifunction Input 4
4	Hex	10	Inverter multifunction Input 5
5	Hex	20	Inverter multifunction Input 6
6	Hex	40	Inverter multifunction Input 7
8	Hex	100	External fault
9	Hex	200	Fault reset
14	Hex	4000	Fault history data clear

Bit	Value	Command	Description
15	Hex	8000	External BB command

- **alarm_number**
  The number of the alarm to read. See the Inverter manual.
- **from**
  The start address of the input to read.
- **length**
  The length of the input to read.

Example No example.

See also N/A

4-2-142 INVERTER_WRITE

Type System command

Syntax **INVERTER_WRITE(0, station, 0, param_number, param_size, VR, mode)**
INVERTER_WRITE(0, station, 2, value)
INVERTER_WRITE(0, station, 3, value)

Description **INVERTER_WRITE** writes the parameter, speed reference or torque reference from the Inverter connected to the system via the MECHATROLINK-II bus.
There are three **INVERTER_WRITE** functions:
- 0: Writes an Inverter parameter.
- 2: Writes the speed reference.
- 3: Writes the torque reference.

To use an Inverter via MECHATROLINK-II you should put the command and the reference via communication option:
- Inverter MV/V7: N3=3; N4=9
- Inverter F7/G7: B1-01=3; B1-02=3.

Make you sure that the Inverter firmware supports the MECHATROLINK-II board.
The command returns -1 if successfully executed and 0 if failed. The result (if any) is returned in the selected VR.

Arguments
- **station**
  The MECHATROLINK-II station number of the Inverter
- **param_number**
  The number of the parameter to write. See the Inverter manual.
- **param_size**
  The size of the parameter to write, 2 or 4 bytes. Most of the Inverter parameters are 2 bytes long. See the Inverter manual.
- **VR**
  The address in the VR memory of the CJ1W-MCH72 where the new value for the parameter is.
- **mode**
  0 = just write; 1= write and enter; 2 = write and config.
- **value**
  The new value that is written.

Example
```
>>INVERTER_WRITE(1,$23,2,3500)
>>INVERTER_READ(1,$23,2,100)
>>PRINT VR(100)
3500.0000
```

See also N/A

Note If you have to transfer many parameters at the same time, the most efficient way is to use MODE 0 for all but the last parameter, and MODE 1 for the last parameter. MODE 0 is executed faster than MODE 1.

4-2-143 JOGSPEED

Type Axis parameter

Syntax **JOGSPEED**

Description The **JOGSPEED** parameter sets the jog speed in user units for an axis. A jog will be performed when a jog input for an axis has been declared and that input is low. A forward jog input and a reverse jog input are available for each axis, respectively set by **FWD_JOG** and **REV_JOG**. The speed of the jog can be controlled with the **FAST_JOG** input.

Arguments N/A

Example No example.

See also **AXIS AXIS**, **FAST_JOG**, **FWD_JOG**, **REV_JOG**, **UNITS**.

4-2-144 LAST_AXIS

Type System parameter (read-only)

Syntax **LAST_AXIS**

Description The **LAST_AXIS** parameter contains the number of the last axis processed by the system.
Most systems do not use all the available axes. It would therefore be a waste of time to task the idle moves on all axes that are not in use. To avoid this to some extent, the CJ1W-MCH72 will task moves on the axes from 0 to **LAST_AXIS**, where **LAST_AXIS** is the number of the highest axis for which an **AXIS** or **BASE** command has been processed, whichever of the two is larger.

Arguments N/A

Example No example.

See also **AXIS**, **BACKLASH**.

4-2-145 LINKAX

Type Axis parameter (read-only)

Syntax **LINKAX**

Description Returns the axis number that the axis is linked to during any linked moves. Linked moves are where the demand position is a function of another axis, e.g. **CONNECT**, **CAMBOX** and **MOVELINK**.

Arguments N/A

Example No example.

See also **CONNECT**, **CAMBOX**, **MOVELINK**.

4-2-146 LIST

Type Program command (Trajexia Studio command line only)

Syntax **LIST ["program_name"]**
TYPE ["program_name"]

Description For use only with the Command Line Terminal interface. **LIST** is used as an immediate (command line) command only and must not be used in programs.
The **LIST** command prints the current selected program or the program specified by **program_name**. The program name can also be specified without quotes. If the program name is omitted, the current selected program will be listed.
Note: This command is implemented for an offline Command Line Terminal. Within Trajexia Studio users can use the terminal window.

Arguments
- **program_name**
  The program to be printed.

Example No example.

See also **SELECT**.

4-2-147 LIST_GLOBAL

Type System command (terminal only)

Syntax **LIST_GLOBAL**

Description When executed from the Command Line Terminal interface (channel 0), all the currently set **GLOBAL** and **CONSTANT** parameters will be printed to the terminal.

Arguments N/A

Example In an application where the following GLOBAL and CONSTANT have been set:
CONSTANT "cutter", 23
GLOBAL "conveyor",5

```
>>LIST_GLOBAL
 Global VR
 ---------- ----
 conveyor 5
 Constant Value
 ---------- -------
 cutter 23.0000
```

See also N/A

4-2-148 LN

Type Mathematical function

Syntax **LN(expression)**

Description The **LN** function returns the natural logarithm of the expression. The input expression value must be greater than 0.

Arguments
- **expression**
  Any valid BASIC expression.

Example
```
>> PRINT LN(10)
2.3026
```

See also N/A

4-2-149 LOCK

Type System command

Syntax
```
LOCK(code)
UNLOCK(code)
```

Description The **LOCK** command prevents the program from being viewed, modified or deleted by personnel unaware of the security code. The lock code number is stored in the flash EPROM. The **UNLOCK** command allows the locked state to be unlocked. The code number can be any integer and is held in encoded form. **LOCK** is always an immediate command and can be issued only when the system is **UNLOCKED**.
LOCK is available from within Trajexia Studio, users can lock the device from the **Online** menu.

Arguments
- **code**
  Any valid integer with maximum 7 digits.

Example
```
>> LOCK(561234)
The programs cannot be modified or seen.
>> UNLOCK(561234)
The system is now unlocked.
```

See also N/A

⚠ **Caution** The security code must be remembered; it will be required to unlock the system. Without the security code the system can not be recovered.

4-2-150 MARK

Type Axis parameter (read-only)

Syntax **MARK**

Description The **MARK** is set to **FALSE** when the **REGIST** command has been executed and is set to **TRUE** when the primary registration event occurs. Only when this parameter is **TRUE**, the **REG_POS** value is correct.

Arguments N/A

Example
```
loop:
  WAIT UNTIL IN(punch_clr)=ON
  MOVE(index_length)
  REGIST(3)
  WAIT UNITL(MARK)
  MOVEMODIFY(REG_POS + offset)
  WAIT IDLE
GOTO loop
```

See also **AXIS**, **REGIST**, **REG_POS**.

4-2-151 MARKB

Type Axis parameter (read-only)

Syntax **MARKB**

Description The **MARKB** is set to **FALSE** when the **REGIST** command has been executed and is set to **TRUE** when the primary registration event occurs. Only when this parameter is **TRUE**, the **REG_POS**B value is correct.

Arguments N/A

Example
```
IF MARKB AXIS(2) THEN
  PRINT "Secondary registration event for axis 2 occurred"
ENDIF
```

See also **AXIS**, **REGIST**, **REG_POSB**.

4-2-152 MECHATROLINK

Type System command

Syntax **MECHATROLINK(0,0)**
Detects and connects slaves on the MECHATROLINK-II connection. It is necessary to use it to reset the network from a communication problem and to re-detect servos that have been not detected (EG: when the A letter in the **AXISSTATUS** word becomes capital red).
MECHATROLINK(0,3,VR)
Returns the number of detected MECHATROLINK-II slaves after a **MECHATROLINK(0,0)**. It is used by the **STARTUP** program to check that the number of detected MECHATROLINK-II stations corresponds with the expected.
MECHATROLINK(0,4,station,VR)
Returns the address of MECHATROLINK-II slave at that "station" number. The station numbers are a sequence 0..x for all the attached slaves. -1 is returned if no slave is allocated to that station. It is used by the **STARTUP** program to check that the number of detected MECHATROLINK-II stations corresponds with the expected.
MECHATROLINK(0,5,station,VR)
Reads and clears missed message count. A Non-Axis MECHATROLINK-II slave does not report automatically a network problem so, use this command to poll the Inverter for checking that the network is alive.
Note:

- You can use the command **MECHATROLINK(0,5,station,VR)** to monitor the status of a slave during a program execution.
  If the contents of the VR address is greater than 0 a communication error with the slave occurs and the slave can malfunction. You can use this command to stop your program when the slave has an error.

Description Note: This command has two forms, depending upon the function required: Master and Station Functions.
All **MECHATROLINK** functions return **TRUE** (-1) if the command was successful or **FALSE** (0) if the command failed.
The functions are separated out into 2 types, **Master** functions that work on a unit, and **Station** functions that work on a specific **station_address** of a given unit.
All functions that retrieve a value store it in the **VR** variable indicated in the last parameter. If this parameter has the value -1 then the value is printed to the command line port.

Arguments N/A

Example No example.

See also N/A

4-2-153 MERGE

Type Axis parameter

Syntax **MERGE**

Description The **MERGE** parameter is a software switch that can be used to enable or disable the merging of consecutive moves. When **MERGE** is ON and the next move already in the next move buffer (**NTYPE**), the axis will not ramp down to 0 speed but will load up the following move enabling a seamless merge. The default setting of **MERGE** is OFF.
It is up to the programmer to ensure that merging is sensible. For example, merging a forward move with a reverse move will cause an attempted instantaneous change of direction.
MERGE will only function if the following are all true:

1 Only the speed profiled moves **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL**, **REVERSE**, **FORWARD** and **MOVEMODIFY** can be merged with each other. They cannot be merged with linked moves **CONNECT**, **MOVELINK** and **CAMBOX**.
2 There is a move in the next move buffer (**NTYPE**).
3 The axis group does not change for multi-axis moves.

When merging multi-axis moves, only the base axis **MERGE** axis parameter needs to be set.
Note: If the moves are short, a high deceleration rate must be set to avoid the CJ1W-MCH72 decelerating in anticipation of the end of the buffered move.

Arguments N/A

Example
```
MERGE = OFF ' Decelerate at the end of each move
MERGE = ON ' Moves will be merged if possible
```

See also **AXIS**.

4-2-154 MHELICAL

Type Axis command

Syntax **MHELICAL(end1, end2, centre1, centre2, direction, distance3 [,mode]))**
MH(end1, end2, centre1, centre2, direction, distance3 [,mode])

Description

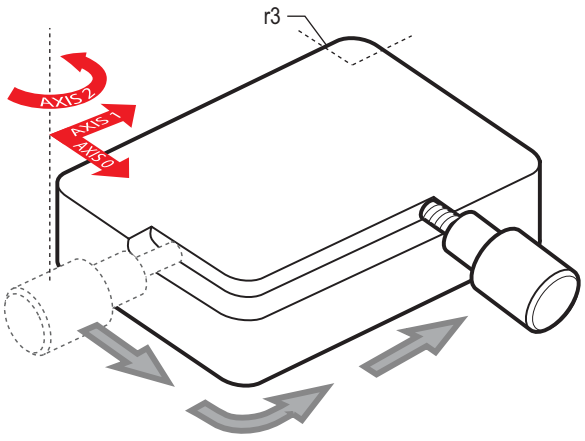
Performs a helical move, that is, moves 2 orthogonal axes in such a way as to produce a circular arc at the tool point with a simultaneous linear move on a third axis. The first 5 parameters are similar to those of a **MOVECIRC** command. The sixth parameter defines the simultaneous linear move.

Finish 1 and centre 1 are on the current **BASE** axis. Finish 2 and centre 2 are on the following axis.

The first 4 distance parameters are scaled according to the current unit conversion factor for the BASE axis. The sixth parameter uses its own axis units.

Arguments

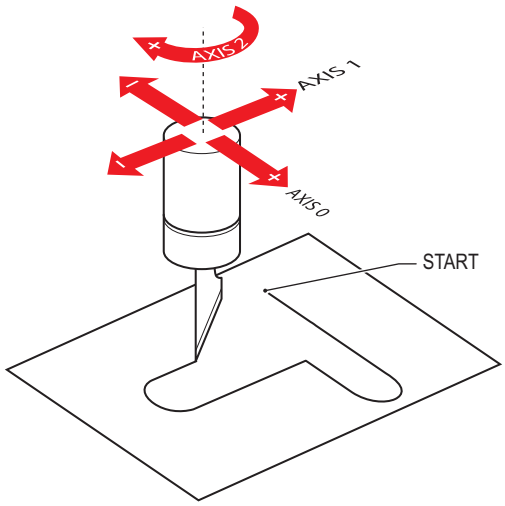
- **end1**
  Position on **BASE** axis to finish at.
- **end2**
  Position on next axis in **BASE** array to finish at.
- **centre1**
  Position on **BASE** axis about which to move.
- **centre2**
  Position on next axis in **BASE** array about which to move.
- **direction**
  The **direction** is a software switch which determines whether the arc is interpolated in a clockwise or anti- clockwise direction. The parameter is set to 0 or 1. See **MOVECIRC**.
- distance3
  The distance to move on the third axis in the **BASE** array axis in user units.
- mode
  0 = Interpolate the third axis with the main two axis when calculating path speed (true helical path).
  1 = Interpolate only the first two axes for path speed, but move the third axis in coordination with the other 2 axes (circular path with following third axis).



Example

The command sequence follows a rounded rectangle path with axis 1 and 2. Axis 3 is the tool rotation so that the tool is always perpendicular to the product. The **UNITS** for axis 3 are set such that the axis is calibrated in degrees.

```
REP_DIST AXIS(3)=360
REP_OPTION AXIS(3)=ON
' all 3 axes must be homed before starting
MERGE=ON
MOVEABS(360) AXIS(3) point axis 3 in correct starting direction
WAIT IDLE AXIS(3)
MOVE(0,12)
MHELICAL(3,3,3,0,1,90)
MOVE(16,0)
MHELICAL(3,-3,0,-3,1,90)
MOVE(0,-6)
MHELICAL(-3,-3,-3,0,1,90)
MOVE(-2,0)
MHELICAL(-3,3,0,3,1,90)
```



Example A PVC cutter uses 2 axes similar to a X-Y plotter. The third axis is used to control the cutting angle of the knife. To keep the resultant cutting speed for the x and y axis equal when cutting curves, mode 1 is applied to the helical command.

```
BASE(0,1,2) : MERGE=ON
'merge moves into one continuous movement
MOVE(50,0)
MHELICAL(0,-6,0,-3,1,180,1)
MOVE(-22,0)
WAIT IDLE
MOVE(-90) AXIS(2) 'rotate the knife after stopping at corner
WAIT IDLE AXIS(2)
MOVE(0,-50)
MHELICAL(-6,0,-3,0,1,180,1)
MOVE(0,50)
WAIT IDLE 'pause again to rotate the knife
MOVE(-90) AXIS(2)
WAIT IDLE AXIS(2)
MOVE(-22,0)
MHELICAL(0,6,0,3,1,180,1)
WAIT IDLE
```

See also **MOVECIRC**.

4-2-155 MOD

Type Mathematical function

Syntax **expression1 MOD expression2**

Description The **MOD** function returns the **expression2** modulus of **expression1**. This function will take the integer part of any non-integer input.

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example
```
>> PRINT 122 MOD 13
5.0000
```

See also N/A

4-2-156 MOTION_ERROR

Type System parameter (read-only)

Syntax **MOTION_ERROR**

Description The **MOTION_ERROR** parameter contains a bit pattern showing the axes which have a motion error. For example. if axis 2 and 6 have the motion error the MOTION_ERROR value would be 68 (4+64).
A motion error occurs when the **AXISSTATUS** state for one of the axes matches the **ERRORMASK** setting. In this case the enable switch (**WDOG**) will be turned off, and **MOTION_ERROR** contains a bit pattern showing all axes which have the motion error. The **ERROR_AXIS** parameter will contain the number of the first axis to have the error.
A motion error can be cleared executing a **DATUM(0)** command or resetting the controller with an EX command.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **DATUM**, **ERROR_AXIS**, **ERRORMASK**, **WDOG**.

4-2-157 MOVE

Type Axis command

Syntax **MOVE(distance_1 [, distance_2 [, distance_3 [, distance_4 [, ...]]]])**
MO(distance_1 [, distance_2 [, distance_3 [, distance_4 [, ...]]]])

Description The **MOVE** command moves with one or more axes at the demand speed and acceleration and deceleration to a position specified as increment from the current position. In multi-axis moves the movement is interpolated and the speed, acceleration and deceleration are taken from the base axis.
The specified distances are scaled using the unit conversion factor in the **UNITS** axis parameter. If, for example, an axis has 4,000 encoder edges/mm, then the number of units for that axis would be set to 4000, and **MOVE**(12.5) would move 12.5 mm.
MOVE works on the default basis axis group (set with **BASE**) unless **AXIS** is used to specify a temporary base axis. Argument **distance_1** is applied to the base axis, **distance_2** is applied to the next axis, etc. By changing the axis between individual **MOVE** commands, uninterpolated, unsynchronised multi-axis motion can be achieved. Incremental moves can be merged for profiled continuous path movements by turning on the **MERGE** axis parameter.
Considering a 2-axis movement, the individual speeds are calculated using the equations below. Given command **MOVE(**x_1,x_2**)** and the profiled speed v_p as calculated from the **SPEED**, **ACCEL** and **DECEL** parameters from the base axis and the total multi-axes distance L = **SQR(**$x_1^2 + x_2^2$**)**.
The individual speed v_i for axis i at any time of the movement is calculated as: $v_i = (x_i * v_p) / L$.

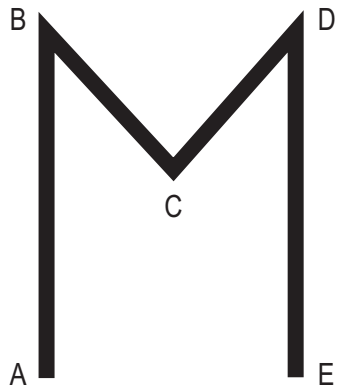
Arguments The command can take up to 16 arguments.

- **distance_i**
  The distance to move for every axis in user units starting with the base axis.

Example A system works with a unit conversion factor of 1 and has a 1000 line encoder. Note that a 1000 line encoder gives 4000 edges/turn.
MOVE(40000) ' move 10 turns on the motor.

Example Axes 3, 4 and 5 must move independently, that is, without interpolation. Each axis moves at its own programmed **SPEED**, **ACCEL** and **DECEL** etc.

```
'setup axis speed and enable
BASE(3)
SPEED=5000
ACCEL=100000
DECEL=150000
SERVO=ON
BASE(4)
SPEED=5000
ACCEL=150000
DECEL=560000
SERVO=ON
BASE(5)
SPEED=2000
ACCEL=320000
DECEL=352000
SERVO=ON
WDOG=ON
MOVE(10) AXIS(5) 'start moves
MOVE(10) AXIS(4)
MOVE(10) AXIS(3)
WAIT IDLE AXIS(5) 'wait for moves to finish
WAIT IDLE AXIS(4)
WAIT IDLE AXIS(3)
```



Example An X-Y plotter can write text at any position within its working envelope. Individual characters are defined as a sequence of moves relative to a start point. Therefore, the same commands can be used regardless of the plot origin. The command subroutine for the letter M is:

```
write_m:
  MOVE(0,12) 'move A > B
  MOVE(3,-6) 'move B > C
  MOVE(3,6) 'move C > D
  MOVE(0,-12)'move D > E
RETURN
```

See also **AXIS**, **MOVEABS**, **UNITS**.

4-2-158 MOVEABS

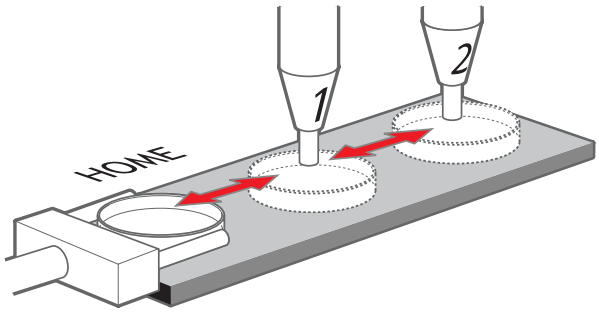
Type Axis command

Syntax **MOVEABS(distance_1 [, distance_2 [, distance_3 [, distance_4 [, ...]]]])**
MA(distance_1 [, distance_2 [, distance_3 [, distance_4 [, ...]]]])

Description The **MOVEABS** command moves one or more axes at the demand speed, acceleration and deceleration to a position specified as absolute position, i.e., in reference to the origin. In multi-axis moves the movement is interpolated and the speed, acceleration and deceleration are taken from the base axis.
The specified distances are scaled using the unit conversion factor in the **UNITS** axis parameter. If, for example, an axis has 4,000 encoder edges/mm, then the number of units for that axis would be set to 4000, and **MOVEABS**(12.5) would move to a position 12.5 mm from the origin.**MOVEABS** works on the default basis axis group (set with **BASE**) unless **AXIS** is used to specify a temporary base axis. Argument **distance_1** is applied to the base axis, **distance_2** is applied to the next axis, etc. By changing the axis between individual **MOVE** commands, uninterpolated, unsynchronised multi-axis motion can be achieved. Absolute moves can be merged for profiled continuous path movements by turning on the **MERGE** axis parameter.
Considering a 2-axis movement, the individual speeds are calculated using the equations below. Given command **MOVE(**ax_1,ax_2**)**, the current position (ay_1,ay_2) and the profiled speed v_p as calculated from the **SPEED**, **ACCEL** and **DECEL** parameters from the base axis and the total multi-axes distance L = **SQR(**$x_1^2 + x_2^2$**)**, where $x_1 = ax_i - ay_i$.
The individual speed for axis at any time of the movement is calculated as $v_i = (x_i \times v_p) / L$.

Arguments The command can take up to 16 arguments.

- **distance_i**
  The position to move every axis *i* to in user units starting with the base axis.



Example

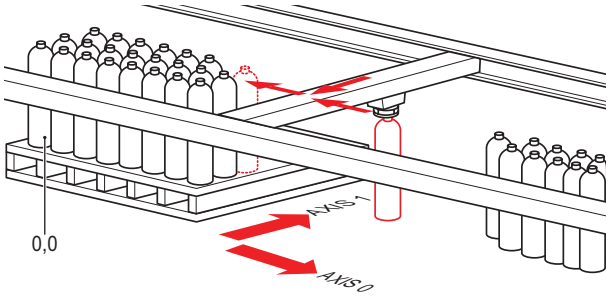
A machine must move to one of 3 positions depending on the selection made by 2 switches. The options are home (if both switches are off), position 1 (if the first switch is on and the second switch is off) and position 2 (if the first switch is off and the second switch is on). Position 2 has priority over position 1.

```
'define absolute positions
home=1000
position_1=2000
position_2=3000
WHILE IN(run_switch)=ON
 IF IN(6)=ON THEN 'switch 6 selects position 2
  MOVEABS(position_2)
  WAIT IDLE
 ELSEIF IN(7)=ON THEN 'switch 7 selects position 1
  MOVEABS(position_1)
  WAIT IDLE
 ELSE
  MOVEABS(home)
  WAIT IDLE
 ENDIF
WEND
```

Example

An X-Y plotter has a pen carousel. The position of this carousel is fixed relative to the absolute zero position of the plotter. To change pens, an absolute move to the carousel position finds the target irrespective of the plot position.

```
MOVEABS(28.5,350) ' move to just outside the pen holder area
WAIT IDLE
SPEED = pen_pickup_speed
MOVEABS(20.5,350) ' move in to pick up the pen
```



Example A pallet consists of a 6 by 8 grid in which gas canisters are inserted 185 mm apart by a packaging machine. The canisters are picked up from a fixed point. The first position in the pallet is defined as position 0,0 with the **DEFPOS** command. The part of the program to position the canisters in the pallet is:

```
FOR x=0 TO 5
 FOR y=0 TO 7
  MOVEABS(-340,-516.5) 'move to pick-up point
  WAIT IDLE
  GOSUB pick 'call pick up subroutine
  PRINT Move to Position: ;x*6+y+1
  MOVEABS(x*185,y*185) 'move to position in grid
  WAIT IDLE
  GOSUB place 'call place down subroutine
 NEXT y
NEXT x
```

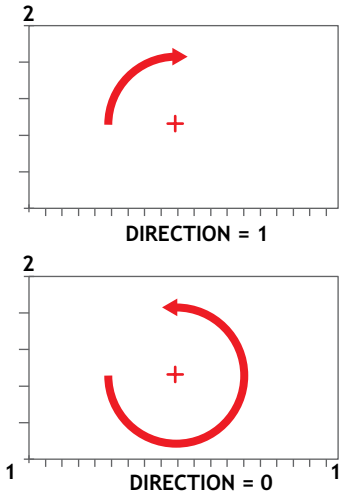
See also **AXIS**, **MOVE**, **MOVEABS**, **UNITS**.

4-2-159 MOVECIRC

Type Axis command

Syntax **MOVECIRC(end_1,end_2,centre_1,centre_2,direction)**
MC(end_1,end_2,centre_1,centre_2,direction)

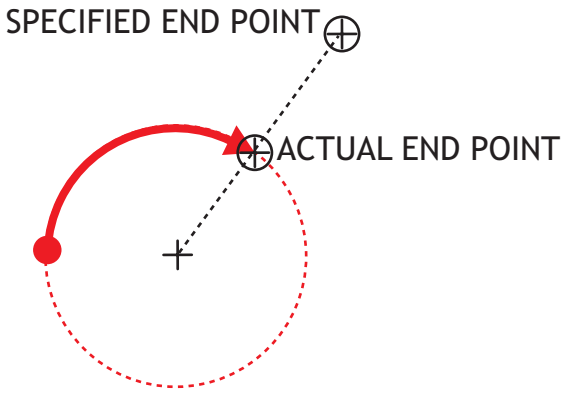
Description The **MOVECIRC** command interpolates 2 orthogonal axes in a circular arc at the tool point. The path of the movement is determined by the 5 arguments, which are incremental from the current position.
The arguments **end_1** and **centre_1** apply to the **BASE** axis and **end_2** and **centre_2** apply to the following axis. All arguments are given in user units of each axis. The speed of movement along the circular arc is set by the **SPEED**, **ACCEL** and **DECEL** parameters of the **BASE** axis. The first four distance parameters are scaled according to the current unit conversion factor for the **BASE** axis.
MOVECIRC works on the default basis axis group (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
For **MOVECIRC** to be correctly executed, the two axes moving in the circular arc must have the same number of encoder pulses per linear axis distance. If they do not, it is possible to adjust the encoder scales in many cases by adjusting with **ENCODER_RATIO** axis parameters for the axis.



Arguments

- **end_1**
  The end position for the **BASE** axis.
- **end_2**
  The end position for the next axis.
- **centre_1**
  The position around which the **BASE** axis is to move.
- **centre_2**
  The position around which the next axis is to move.
- **direction**
  A software switch that determines whether the arc is interpolated in a clockwise or counterclockwise direction. Value: 0 or 1.
  If the two axes involved in the movement form a right-hand axis, set direction to 0 to produce positive motion about the third (possibly imaginary) orthogonal axis. If the two axes involved in the movement form a left-hand axis. set direction to 0 to produce negative motion about the third (possibly imaginary) orthogonal axis. See the table below.

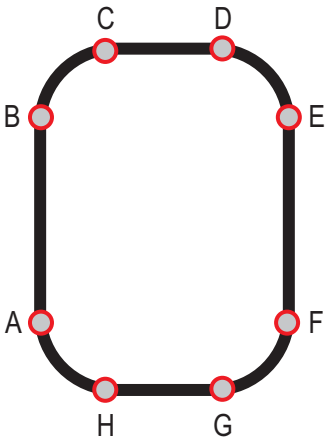
Direction	Right-hand axis	Left-hand axis
0	Positive	Negative
1	Negative	Positive



Note: In order for the **MOVECIRC** to be correctly executed, the two axes generating the circular arc must have the same number of encoder pulses versus linear axis distance. If this is not the case it is possible to adjust the encoder scales in many cases by using **ENCODER_RATIO** parameter.

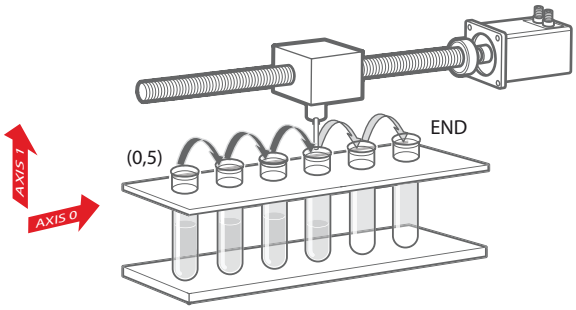
Note: The **MOVECIRC** computes the radius and the total angle of rotation from the centre, and end-point. If the end point is not on the calculated path, the move simply ends at the computed end and not the specified end point. It is the responsibility of the programmer to ensure that the two points correspond to correct points on a circle.

Note: Neither axis may cross the set absolute repeat distance REP_DIST during a **MOVECIRC. Doing so may cause one or both axis to jump or for their FE value to exceed FE_LIMIT.**



Example The following command sequence plots the letter O:

```
MOVE(0,6) ' Move A -> B
MOVECIRC(3,3,3,0,1) ' Move B -> C
MOVE(2,0) ' Move C -> D
MOVECIRC(3,-3,0,-3,1) ' Move D -> E
MOVE(0,-6) ' Move E -> F
MOVECIRC(-3,-3,-3,0,1) ' Move F -> G
MOVE(-2,0) ' Move G -> H
MOVECIRC(-3,3,0,3,1)  ' Move H -> A
```



Example A machine is required to drop chemicals into test tubes. The nozzle can move up and down and also along its rail. The most efficient motion for the nozzle is to move in an arc between the test tubes.

```
BASE(0,1)
MOVEABS(0,5) 'move to position above first tube
MOVEABS(0,0) 'lower for first drop
WAIT IDLE
OP(15,ON) 'apply dropper
WA(20)
OP(15,OFF)
FOR x=0 TO 5
MOVECIRC(5,0,2.5,0,1) 'arc between the test tubes
WAIT IDLE
OP(15,ON) 'Apply dropper
WA(20)
OP(15,OFF)
NEXT x
MOVECIRC(5,5,5,0,1) 'move to rest position
```

See also **AXIS**, **ENCODER_RATIO**, **UNITS**

4-2-160 MOVELINK

Type Axis command

Syntax **MOVELINK(distance, link_distance, link_acceleration, link_deceleration, link_axis [, link_option [, link_position]])**
ML(distance, link_distance, link_acceleration, link_deceleration, link_axis [, link_option [, link_position]])

Description

The **MOVELINK** command creates a linear move on the base axis linked via a software gearbox to the measured position of a link axis. The link axis can move in either direction to drive the output motion.

The parameters show the distance the BASE axis moves for a certain distance of the link axis (**link_distance**). The link axis distance is divided into three phases that apply to the movement of the base axis. These parts are the acceleration, the constant speed and the deceleration. The link acceleration and deceleration distances are specified by the **link_acceleration** and **link_deceleration** parameters. The constant speed link distance is derived from the total link distance and these two parameters.

The three phases can be divided into separate **MOVELINK** commands or can be added up together into one.

Consider the following two rules when setting up the **MOVELINK** command.

Rule 1: In an acceleration and deceleration phase with matching speed, the **link_distance** must be twice the distance. See the figure.

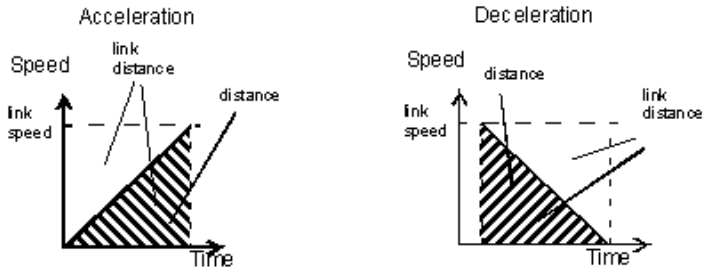
Rule 2: In a constant speed phase with matching speeds, the two axes travel the same distance so the distance to move must equal the **link_distance**.

MOVELINK works on the default basis axis group (set with **BASE**) unless **AXIS** is used to specify a temporary base axis. The axis set for **link_axis** drives the base axis.

MOVELINK is designed for controlling movements such as:

- Synchronization to conveyors
- Flying shears
- Thread chasing, tapping etc.
- Coil winding

Note: If the sum of **link_acceleration** and **link_deceleration** is greater than **link_distance**, they are both reduced in proportion in order to equal the sum to **link_distance**.



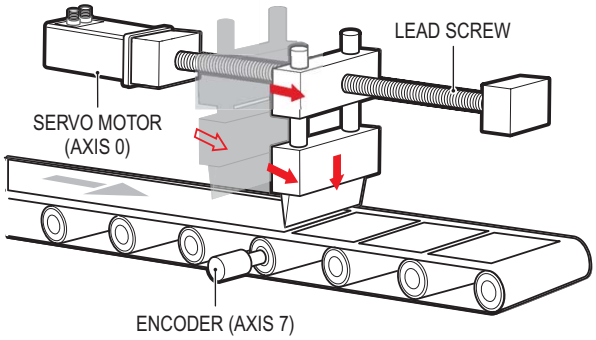
Arguments

- **distance**
  The incremental distance in user units to move the BASE axis, as a result of the measured **link_distance** movement on the link axis.
- **link_distance**
  The positive incremental distance in user units that is required to be measured on the link axis to result in the distance motion on the BASE axis.
- **link_acceleration**
  The positive incremental distance in user units on the link axis over which the base axis will accelerate.
- **link_deceleration**
  The positive incremental distance in user units on the link axis over which the base axis will decelerate.
  Note: If the sum of parameter 3 and parameter 4 is greater than parameter 2, they are both reduced in proportion until their sum equals parameter 2.
- **link axis**
  The axis to link to.
- **link_option**
  See the table below.

link_option	Description
1	Link starts when registration event occurs on link axis.
2	Link starts at an absolute position on link axis (see **link_position**).
4	**MOVELINK** repeats automatically and bidirectionally. This option is cancelled by setting bit 1 of **REP_OPTION** parameter (that is, **REP_OPTION = REP_OPTION OR 2**).
5	Combination of options 1 and 4.
6	Combination of options 2 and 4.

- **link_position**
  The absolute position where **MOVELINK** will start when **link_option** is set to 2

Note: The command uses the BASE and AXIS, and unit conversion factors in a similar way to other MOVE commands.
Note: The “link” axis may move in either direction to drive the output motion. The link distances specified are always positive.



Example

A flying shear that moves at the speed of the material cuts a long sheet of paper into cards every 160 m. The shear can move up to 1.2 metres, of which 1m is used in this example. The paper distance is measured by an encoder. The unit conversion factor is set to give units of metres on both axes. Note that axis 7 is the link axis.

```
WHILE IN(2)=ON
  MOVELINK(0,150,0,0,7) ' dwell (no movement) for 150m
  MOVELINK(0.3,0.6,0.6,0,7) ' accelerate to paper speed
  MOVELINK(0.7,1.0,0,0.6,7) ' track the paper then decelerate
  WAIT LOADED ' wait until acceleration movelink is finished
  OP(8,ON) ' activate cutter
  MOVELINK(-1.0,8.4,0.5,0.5,7)  retract cutter back to start
  WAIT LOADED
  OP(8,OFF) ' deactivate cutter at end of outward stroke
WEND
```

In this program, the controller waits for the roll to feed out 150 m in the first

line. After this distance the shear accelerates to match the speed of the paper, moves at the same speed, and then decelerates to a stop within the 1 m stroke. This movement is specified using two separate **MOVELINK** commands. This allows the program to wait for the next move buffer to be clear, **NTYPE=0**, which indicates that the acceleration phase is complete. Note that the distances on the measurement axis (the link distance in each **MOVELINK** command), 150, 0.8, 1.0 and 8.2, add up to 160 m. To make sure that the speed and the positions of the cutter and paper match during the cut process, the parameters of the **MOVELINK** command must be correct. The easiest way to do this is to consider the acceleration, constant speed and deceleration phases separately, and then combine them as required, according to these 2 rules:

Rule 1: In an acceleration phase to a matching speed, the link distance must be twice the movement distance. Therefore, the acceleration phase can be specified alone as:

MOVELINK(0.3,0.6,0.6,0,1)' move is all accel

Rule 2: In a constant speed phase with matching speed, the two axes move the same distance. Therefore, the distance to move must be equal the link distance. Therefore, the constant speed phase can be specified as:

MOVELINK(0.4,0.4,0,0,1)' all constant speed

The deceleration phase is set in this case to match the acceleration:

MOVELINK(0.3,0.6,0,0.6,1)' all decel

The movements of each phase can be added to give the total movement.

MOVELINK(1,1.6,0.6,0.6,1)' Same as 3 moves above

But in the example above, the acceleration phase is kept separate:

MOVELINK(0.3,0.6,0.6,0,1)

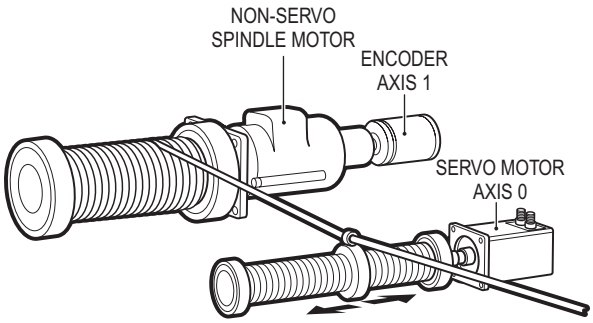
MOVELINK(0.7,1.0,0,0.6,1)

This allows the output to be switched on at the end of the acceleration phase.

Example

MOVELINK can be used to create an exact ratio gearbox between two axes. Suppose it is required to create a gearbox link of 4000/3072. This ratio is inexact (1.30208333). If this ratio is entered into a **CONNECT** command, the axes will slowly creep out of synchronisation. To prevent this problem, set the “link option” to 4 to make **MOVELINK** repeat continuously.

MOVELINK(4000,3072,0,0,linkaxis,4)



Example In this example on coil winding the unit conversion factors **UNITS** are set so that the payout movements are in mm and the spindle position is measured in revolutions. The payout eye therefore moves 50 mm over 25 revolutions of the spindle with the command **MOVELINK(50,25,0,0,linkax)**. To accelerate over the first spindle revolution and decelerate over the final 3 use the command **MOVELINK(50,25,1,3,linkax)**.

```
OP(motor,ON) ' Switch spindle motor on
FOR layer=1 TO 10
  MOVELINK(50,25,0,0,1)
  MOVELINK(-50,25,0,0,1)
NEXT layer
WAIT IDLE
OP(motor,OFF)
```

See also **AXIS**, **UNITS**, **REP_OPTION**.

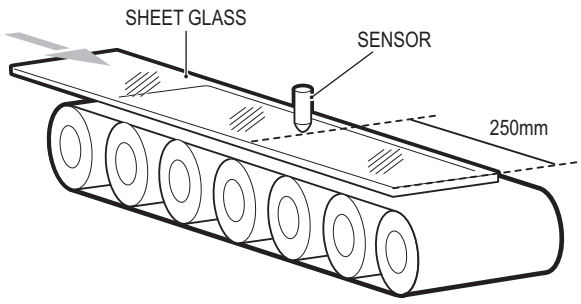
4-2-161 MOVEMODIFY

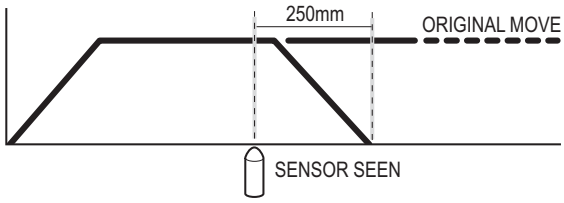
Type Axis command

Syntax **MOVEMODIFY(position)**
MM(position)

Description The **MOVEMODIFY** command changes the absolute end position of the current single-axis linear move (**MOVE**, **MOVEABS**). If there is no current move or the current move is not a linear move, then **MOVEMODIFY** is treated as a **MOVEABS** command. The **ENDMOVE** parameter will contain the position of the end of the current move in user units.
MOVEMODIFY works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

Arguments
- **position**
  The absolute position to be set as the new end of move.

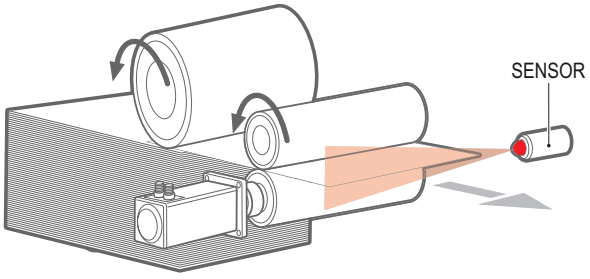




Example

A sheet of glass is fed on a conveyor and is required to stop 250 mm after the leading edge is sensed by a proximity switch. The proximity switch is connected to the registration input:

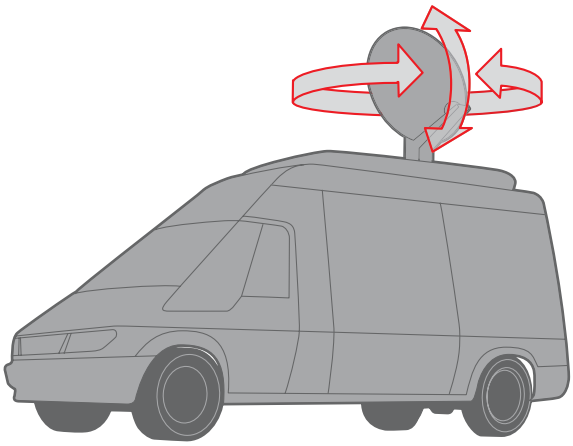
```
MOVE(10000) 'Start a long move on conveyor
REGIST(3) 'set up registration
WAIT UNTIL MARK
'MARK becomes TRUE when sensor detects glass edge
OFFPOS = -REG_POS 'set position where mark was seen to 0
WAIT UNTIL OFFPOS=0 'wait for OFFPOS to take effect
MOVEMODIFY(250) 'change move to stop at 250mm
```



Example

A paper feed system slips. To counteract this, a proximity sensor is positioned one third of the way into the movement. This detects at which position the paper passes, and thus how much slip has occurred. The move is then modified to account for this variation.

```
paper_length=4000
DEFPOS(0)
REGIST(3)
MOVE(paper_length)
WAIT UNTIL MARK
slip=REG_POS-(paper_length/3)
offset=slip*3
MOVEMODIFY(paper_length+offset)
```



Example

A satellite receiver sits on top of a van. It must align correctly to the satellite from data processed in a computer. This information is sent to the controller through the serial link and sets VR(0) and VR(1). This information is used to control the two axes.
MOVEMODIFY is used so that the position can be continuously changed even if the previous set position is not achieved.

```
bearing=0 'set lables for VRs
elevation=1
UNITS AXIS(0)=360/counts_per_rev0
UNITS AXIS(1)=360/counts_per_rev1
WHILE IN(2)=ON
  MOVEMODIFY(VR(bearing))AXIS(0) 'adjust bearing to match VR0
    MOVEMODIFY(VR(elevation))AXIS(1)'adjust elevation to match VR1
  WA(250)
WEND
RAPIDSTOP 'stop movement
WAIT IDLE AXIS(0)
MOVEABS(0) AXIS(0) 'return to transport position
WAIT IDLE AXIS(1)
MOVEABS(0) AXIS (1)
```

4-2-162 MPOS

Type — Axis parameter (read-only)

Syntax — **MPOS**

Description — The **MPOS** parameter is the measured position of the axis in user units as derived from the encoder. This parameter can be set using the **DEFPOS** command. The **OFFPOS** axis parameter can also be used to shift the origin point. **MPOS** is reset to 0 at start-up or after the controller has been reset.
The range of the measured position is controlled with the **REP_DIST** and **REP_OPTION** axis parameters.

Arguments — N/A

Example **WAIT UNTIL MPOS >= 1250**
SPEED = 2.5

See also **UNITS**, **AXIS**, **DEFPOS**, **ENCODER**, **FE**, **OFFPOS**, **REP_DIST**, **REP_OPTION**, **UNITS**.

4-2-163 MSPEED

Type Axis parameter (read-only)

Syntax **MSPEED**

Description The **MSPEED** parameter contains the measured speed in units/s. It is calculated by taking the change in the measured position in user units in the last servo period and divide it by the servo period (in seconds). The servo period is set with the **SERVO_PERIOD** parameter.
MSPEED represents a snapshot of the speed and significant fluctuations, which can occur, particularly at low speeds. It can be worthwhile to average several readings if a stable value is required at low speeds.

Arguments N/A

Example No example.

See also **AXIS**, **SERVO_PERIOD**, **VP_SPEED**, **UNITS**.

4-2-164 MTYPE

Type Axis parameter (read-only)

Syntax **MTYPE**

Description The **MTYPE** parameter contains the type of move currently being executed. The possible values are given in the table below.

Move number	Move type
0	**IDLE** (no move)
1	**MOVE**
2	**MOVEABS**
3	**MHELICAL**
4	**MOVECIRC**
5	**MOVEMODIFY**
10	**FORWARD**
11	**REVERSE**
12	**DATUM**
13	**CAM**
14	JOG_FORWARD refer to **FWD_JOG**
15	JOG_REVERSE refer to **REV_JOG**
20	**CAMBOX**
21	**CONNECT**
22	**MOVELINK**

MTYPE can be used to determine whether a move has finished or if a transition from one move type to another has taken place.
A non-idle move type does not necessarily mean that the axis is actually moving. It can be at 0 speed part way along a move or interpolating with another axis without moving itself.

Arguments N/A

Example No example.

See also **AXIS**, **NTYPE**.

4-2-165 NEG_OFFSET

Type System parameter

Syntax **NEG_OFFSET=value**

Description For Piezo Operation. This allows a negative offset to be applied to the output **S_REF** signal from the servo loop. An offset of 327 will represent an offset of 0.1 volts. It is suggested that as offset of 65% to 70% of the value required to make the stage move in an open loop situation is used.

Arguments
- **value**
  A BASIC expression.

Example No example.

See also N/A

4-2-166 NEW

Type Program command

Syntax **NEW ["program_name"]**

Description The **NEW** command deletes all program lines of the program in the controller. **NEW** without a program name can be used to delete the currently selected program (using **SELECT**). The program name can also be specified without quotes. **NEW ALL** will delete all programs.
The command can also be used to delete the TABLE memory.
NEW "TABLE" The name **"TABLE"** must be in quotes.
Note: This command is implemented for a Command Line Terminal.

Arguments N/A

Example No example.

See also **COPY**, **DEL**, **RENAME**, **SELECT**, **TABLE**

4-2-167 NEXT

See **FOR..TO..STEP..NEXT**.

4-2-168 NOT

Type Mathematical operation

Syntax **NOT expression**

Description The **NOT** operator performs the logical **NOT** function on all bits of the integer part of the expression.
The logical **NOT** function is defined as in the table below.

Bit	Result
0	1
1	0

Arguments
- **expression.**
  Any valid BASIC expression.

Example
```
>> PRINT 7 AND NOT 1
6.0000
```

See also N/A

4-2-169 NTYPE

Type Axis parameter (read-only)

Syntax **NTYPE**

Description The **NTYPE** parameter contains the type of the move in the next move buffer. Once the current move has finished, the move present in the **NTYPE** buffer will be executed. The values are the same as those for the **MTYPE** axis parameter.
NTYPE is cleared by the **CANCEL(1)** command.

Arguments N/A

Example No example.

See also **AXIS**, **MTYPE**.

4-2-170 OFF

Type Constant (read-only)

Syntax **OFF**

Description The **OFF** constant returns the numerical value 0.

Arguments N/A

Example
```
OP (lever,OFF)
```
The above line sets the output named lever to OFF.

See also N/A

4-2-171 OFFPOS

Type Axis parameter

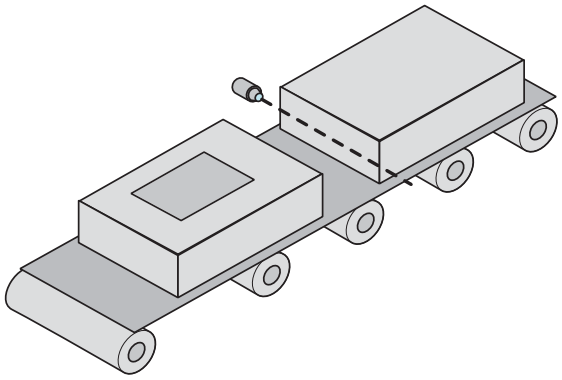
Syntax **OFFPOS**

Description The **OFFPOS** parameter contains an offset that will be applied to the demand position (**DPOS**) without affecting the move which is in progress in any other way. The measured position will be changed accordingly in order to keep the Following Error. OFFPOS can therefore be used to effectively datum a system at full speed. The value set in **OFFPOS** will be reset to 0 by the system as the offset is loaded.
Note: The offset is applied on the next servo period. Other commands may be executed prior to the next servo period. Be sure that these commands do not assume the position shift has occurred. This can be done by using the **WAIT UNTIL** statement (see example).

Arguments N/A

Example Change the current position by 125, with the Command Line Terminal:
>>?DPOS
300.0000
>>OFFPOS=125
>>?DPOS
425.0000

Example Define the current demand position as 0
OFFPOS=-DPOS
WAIT UNTIL OFFPOS=0 ' wait until applied
This is equivalent to **DEFPOS(0)**.



Example A conveyor transports boxes. Labels must be applied onto these boxes. The **REGIST** function can capture the position at which the leading edge of the box is seen. Then, the **OFFPOS** command can adjust the measured position of the axis to make it 0 at that point. Thus, after the registration event has occurred, the measured position (seen in **MPOS**) reflects the absolute distance from the start of the box. The mechanism that applies the label can take advantage of the absolute position start mode of the **MOVELINK** or **CAMBOX** commands to apply the label.
BASE(conv)
REGIST(3)
WAIT UNTIL MARK
OFFPOS = -REG_POS ' Leading edge of box is now zero

See also **AXIS**, **DEFPOS**, **DPOS**, **MPOS**, **UNITS**.

4-2-172 ON

Type	Constant (read-only)
Syntax	**ON**
Description	The **ON** constant returns the numerical value 1.
Arguments	N/A
Example	**OP (lever,ON)** The above line sets the output named lever to **ON**.
See also	N/A

4-2-173 ON.. GOSUB

Type	Program control command
Syntax	**ON expression GOSUB label [,label[,...]]**
Description	The **ON..GOSUB** and **ON..GOTO** structures enable a conditional jump. The integer expression is used to select a label from the list. If the expression has value 1 the first label is used, for value 2 the second label is used, and so on. Once the label is selected, subroutine **GOSUB** jump to that label is performed. Note: If the expression is not valid e.g. the result of the expression is less than 1 or greater that the number of available labels in the program, no jump is performed.
Arguments	• **expression** Any valid BASIC expression. • **label** Any valid label in the program.
Example	**REPEAT** **GET#5,char** **UNTIL 1<=char and char<=3** **ON char GOSUB mover, stopper, change**
See also	**GOSUB..RETURN**, **GOTO**.

4-2-174 ON.. GOTO

Type	Program control command
Syntax	**ON expression GOTO label [,label[,...]]**
Description	The expression is evaluated and then the integer part is used to select a label from the list. If the expression has the value 1 then the first label is used, 2 then the second label is used, and so on. If the value of the expression is less than 1 or greater than the number of labels then an error occurs. Once the label is selected, subroutine **GOTO** jump to that label is performed.
Arguments	• **expression** Any valid BASIC expression. • **label** Any valid label in the program.

Example
```
REPEAT
  GET #1,char
UNTIL 1<=char and char<=3
ON char GOTO mover,stopper,change
```

See also N/A

4-2-175 OP

Type I/O command

Syntax
```
OP(output_number, value)
OP(binary_pattern)
OP
```

Description The **OP** command sets one or more outputs or returns the state of the first 24 outputs. **OP** has three different forms depending on the number of arguments.

- Command **OP(output_number,value)** sets a single output channel. The range of **output_number** depends on the number of additional digital I/O connected to the PLC and **value** is the value to be output, either 0 or 1.
- Command **OP(binary_pattern)** sets the binary pattern to the 24 outputs according to the value set by **binary_pattern**.
- Function **OP (without arguments)** returns the status of the first 24 outputs. This allows multiple outputs to be set without corrupting others which are not to be changed.

Note: The first 8 outputs (0 to 7) do not physically exist on the CJ1W-MCH72. They can not be written to and will always return 0.

Arguments

- **output_number**
  The number of the output to be set. The range for this parameter depends on the number of additional digital I/O connected to the PLC. If there are no digital I/O connected, the range for this parameter is 8..31
- **value**
  The value to be output, either OFF (0) or ON (1). All non-0 values are considered as ON.
- **binary_pattern**
  The integer equivalent of the binary pattern is to be output.

Example
```
OP(12,1)
OP(12,ON)
```
These two lines are equivalent.

Example
```
OP(18*256)
```
This line sets the bit pattern 10010 on the first 5 physical outputs, outputs 13 to 17 would be cleared. The bit pattern is shifted 8 bits by multiplying by 256 to set the first available outputs as outputs 0 to 7 do not exist.

Example
```
VR(0) = OP
VR(0) = VR(0) AND 65280
OP(VR(0))
```
This routine sets outputs 8 to 15 **ON** and all others off.
The above programming can also be written as follows:
```
OP(OP AND 65280)
```

Example
```
val = 8 ' The value to set
mask = OP AND NOT(15*256) ' Get current status and mask
OP(mask OR val*256) ' Set val to OP(8) to OP(11)
```
This routine sets value **val** to outputs 8 to 11 without affecting the other outputs by using masking.

See also **IN**.

4-2-176 OPEN_WIN

Type Axis parameter

Syntax **OPEN_WIN**
OW

Description The **OPEN_WIN** parameter defines the beginning of the window inside or outside which a registration event is expected. The value is in user units.

Arguments N/A

Example
```
only look for registration marks between 170 and 230
OPEN_WIN = 170
CLOSE_WIN = 230
REGIST(256+3)
WAIT UNTIL MARK
```

See also **CLOSE_WIN**, **REGIST**, **UNITS**.

4-2-177 OR

Type Mathematical operation

Syntax **expression1 OR expression2**

Description The **OR** operator performs the logical **OR** function between corresponding bits of the integer parts of two valid BASIC expressions.
The logical **OR** function between two bits is defined as in the table below.

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	1

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **result = 10 OR (2.1*9)**
The parentheses are evaluated first, but only the integer part of the result, 18, is used for the operation. Therefore, this expression is equivalent to the following:
result = 10 OR 18
The **OR** is a bit operator and so the binary action taking place is:
01010 OR 10010 = 11010
Therefore, result will contain the value 26.

Example **IF VR(0) = 1 OR VR(0) = 2 THEN GOTO label**

See also N/A

4-2-178 OUTLIMIT

Type Axis parameter

Syntax **OUTLIMIT**

Description The output limit restricts the demand output from a servo axis to a lower value than the maximum. The value required varies depending on the maximum demand output possible. If the voltage output is generated by a 16 bit S_REF value an **OUTLIMIT** of 32767 will produce the full +/-10v range. A MECHATROLINK-II speed axis has a 32 bit maximum demand.

Arguments N/A

Example **OUTLIMIT AXIS(1) = 16384**
The above will limit the voltage output to a +/-5V output range from the Encoder Interface This will apply to the DAC command if SERVO=OFF, or to the voltage output by the servo loop if SERVO=ON.

See also **AXIS**, **S_REF**, **S_REF_OUT**, **SERVO**.

4-2-179 OV_GAIN

Type Axis parameter

Syntax **OV_GAIN**

Description The **OV_GAIN** parameter contains the output velocity gain. The output velocity output contribution is calculated by multiplying the change in measured position with the **OV_GAIN** parameter value. The default value is 0.
Adding NEGATIVE output velocity gain to a system is mechanically equivalent to adding damping. It is likely to produce a smoother response and allow the use of a higher proportional gain than could otherwise be used, but at the expense of higher Following Errors. High values may cause oscillation and produce high Following Errors.
Note: Negative values are normally required for OV_GAIN.
Note: In order to avoid any instability the servo gains should be changed only when the **SERVO** is off.

Arguments N/A

Example No example.

See also **D_GAIN**, **I_GAIN**, **P_GAIN**, **VFF_GAIN**.

4-2-180 P_GAIN

Type	Axis parameter
Syntax	**P_GAIN**
Description	The **P_GAIN** parameter contains the proportional gain. The proportional output contribution is calculated by multiplying the Following Error with the **P_GAIN** parameter value. The default value of **P_GAIN** for MECHATROLINK-II Speed axis (**ATYPE = 41**) is 131072. The default value for Encoder Interface In (**ATYPE = 44**) is 1.0. The proportional gain sets the stiffness of the servo response. Values that are too high will cause oscillation. Values that are too low will cause large Following Errors. Note: In order to avoid any instability the servo gains should be changed only when the **SERVO** is off.
Arguments	N/A
Example	No example.
See also	**D_GAIN**, **I_GAIN**, **OV_GAIN**, **VFF_GAIN**.

4-2-181 PI

Type	Constant (read-only)
Syntax	**PI**
Description	The **PI** constant returns the numerical constant value of approximately 3.14159.
Arguments	N/A
Example	**circum = 100** **PRINT "Radius = "; circum/(2*PI)**
See also	N/A

4-2-182 PLC_EXCHANGE

Type	System command
Syntax	**PLC_EXCHANGE(0, area_code)** **PLC_EXCHANGE(1, area_code, plc_area, plc_start, tj_area, tj_start, total_items)**
Description	**PLC_EXCHANGE(0, ...)** prints the mapping of the PLC memory area specified by **area_code** to the CJ1W-MCH72 memory. The output of this command is: **area_code**, **plc_area**, **plc_start**, **tj_area**, **tj_start**, **total_items**. **PLC_EXCHANGE(1, ...)** configures the mapping of the PLC memory to the CJ1W-MCH72 memory.

Arguments

- **area_code**
  The PLC area. Possible values are:
  - 0100..0107 for PLC output area (8 areas available)
  - 8100..8107 for PLC input area (8 areas available)
- **plc_area**
  The PLC memory area used for data exchange. Possible values are:
  - 01 : CIO
  - 03 : DM
  - 04 : WR
  - 05 : HR
  - 08..14 : EM bank 0..C
- **plc_start**
  The start address in PLC memory.
  Note: The validity depends on **plc_area**.
- **tj_area**
  The CJ1W-MCH72 memory area used for data exchange. Possible values are:
  - 00 : VR 16-bit signed integer
  - 01 : VR 32-bit floating point
  - 02 : IN or OP array, depending on direction
  - 03 : AIN or AOUT array, depending on direction
  - 04 : Axis Status array, only valid if direction is PLC input
- **tj_start**
  The tart address in CJ1W-MCH72 memory.
  Note: The validity depends on **tj_area**.
- **total_items**
  The total number of items (words and dwords) to transfer
  Note: The validity depends on **plc_area** and **tj_area**.

Example No example.

See also **PLC_STATUS**

4-2-183 PLC_STATUS

Type System parameter (read-only)

Syntax **PLC_STATUS(mode)**

Description The **PLC_STATUS(0)** system parameter contains the monitored PLC CPU/PC21 bus status. The status consists of status bits, which definitions are shown in the table below.

Bit number	Description
0	XSRES signal active
1	XSWDTU signal active
7	Cyclic refresh time-out
8	PLC Program Mode flag
9	FALS (Severe Failure) PLC flag
10	FAL (Non-severe Failure) PLC flag
11	LOAD OFF PLC flag

PLC_STATUS(1) returns the elapsed time in milliseconds since the last cyclic refresh.

Arguments
- **mode**
  0 or 1, to return the status or the elapsed time since the last cyclic refresh.

Example No example.

See also **PLC_EXCHANGE**

4-2-184 PMOVE

Type Task parameter (read-only)

Syntax **PMOVE**

Description The **PMOVE** parameter contains the status of the task buffers. The parameter returns **TRUE** if the task buffers are occupied, and **FALSE** if they are empty.
When the task executes a movement command, the task loads the movement information into the task move buffers. The buffers can hold one movement instruction for any group of axes. **PMOVE** will be set to **TRUE** when loading of the buffers has been completed. When the next servo interrupt occurs, the motion generator loads the movement into the next move (**NTYPE**) buffer of the required axes if they are available. When this second transfer has been completed, **PMOVE** is cleared to 0 until another move is executed in the task.
Each task has its own **PMOVE** parameter. Use the **PROC** modifier to access the parameter for a certain task. Without **PROC** the current task will be assumed.

Arguments N/A

Example No example.

See also **NTYPE**, **PROC**.

4-2-185 POS_OFFSET

Type System parameter

Syntax **POS_OFFSET=value**

Description For Piezo Operation. This keyword allows a positive offset to be applied to the output **S_REF** signal from the servo loop. An offset of 327 will represent an offset of 0.1 volts for axis with servo output generated by a 16 bit S_REF. It is suggested that as offset of 65% to 70% of the value required to make the stage move in an open loop situation is used.

Arguments N/A

Example No example.

See also N/A

4-2-186 POWER_UP

Type System parameter

Syntax **POWER_UP**

Description This parameter is used to determine whether or not programs should be read from flash EPROM on power up or software reset (**EX**).
Two values are possible: 0: Use the programs in battery backed RAM; 1: Copy programs from the controllers flash EPROM into RAM.
Programs are individually selected to be run at power up with the **RUNTYPE** command
Notes:
- **POWER_UP** is always an immediate command and therefore cannot be included in programs.
- This value is normally set by Trajexia Studio.

Arguments N/A

Example No example.

See also **EPROM**

4-2-187 PRINT

Type I/O command

Syntax **PRINT [#n,] expression { , expression}**
? [#n,] expression { , expression }

Description The **PRINT** command outputs a series of characters to the communication ports. **PRINT** can output parameters, fixed ASCII strings, and single ASCII characters. By using **PRINT #n**, any port can be selected to output the information to.
Multiple items to be printed can be put on the same line separated by a comma or a semi-colon. A comma separator in the print command places a tab between the printed items. The semi-colon separator prints the next item without any spaces between printed items.
The width of the field in which a number is printed can be set with the use of [w,x] after the number to be printed. The width of the column is given by w and the number of decimal places is given by x. Using only one parameter [x] takes the default width and specifies the number of decimal places to be printed. The numbers are right aligned in the field with any unused leading characters being filled with spaces. If the number is too long, then the field will be filled with asterisks to signify that there was not sufficient space to display the number. The maximum field width allowable is 127 characters.
The backslash **** command can be used to print a single ASCII character.

Arguments
- **n**
  The specified output device. When this argument is omitted, the port is 0 (Terminal window). See the table below.

Value	Description
0	Programming port 0 (default)
5	Trajexia Studio port 0 user channel 5
6	Trajexia Studio port 0 user channel 6
7	Trajexia Studio port 0 user channel 7

- **expression**
  The expression to be printed.

Example **PRINT "CAPITALS and lower case CAN BE PRINTED"**

Example Consider VR(1) = 6 and **variab** = 1.5, the print output will be as follows:
PRINT 123.45, VR(1)-variab
123.4500 4.5000

Example **length:**
PRINT "DISTANCE = ";mpos
DISTANCE = 123.0000
In this example, the semi-colon separator is used. This does not tab into the next column, allowing the programmer more freedom in where the print items are placed.

Example **PRINT VR(1)[4,1]; variab[6,2]**
6.0 1.50

Example **params:**
PRINT "DISTANCE = ";mpos[0];" SPEED = ";v[2];
DISTANCE = 123 SPEED = 12.34

Example **PRINT "ITEM ";total" OF ";limit;CHR(13);**

Example **>> PRINT HEX(15),HEX(-2)**
F FFFFA

See also **$ (HEXADECIMAL INPUT)**

4-2-188 PROC

Type Task command

Syntax **PROC(task_number)**

Description The **PROC** modifier allows a process parameter from a particular process to be read or written. If omitted, the current task will be assumed.

Arguments
- **task_number**
  The number of the task to access.

Example **WAIT UNTIL PMOVE PROC(3)=0**

See also N/A

4-2-189 PROC_STATUS

Type Task parameter

Syntax **PROC_STATUS**

Description The **PROC_STATUS** parameter returns the status of the process or task specified. The parameter is used with the **PROC** modifier and can return values listed in the table below.

Value	Description
0	Process stopped
1	Process running
2	Process stepping
3	Process paused

Arguments N/A

Example	**WAIT UNTIL PROC_STATUS PROC(3)=0**
See also	**PROCNUMBER**, **PROC**.

4-2-190 PROCESS

Type	Program command
Syntax	**PROCESS**
Description	The **PROCESS** command displays the running status of all current tasks with their task number.
Arguments	N/A
Example	No example.
See also	**HALT**, **RUN**, **STOP**.

4-2-191 PROCNUMBER

Type	Task parameter (read-only)
Syntax	**PROCNUMBER**
Description	The **PROCNUMBER** parameter contains the number of the task in which the currently selected program is running. **PROCNUMBER** is often required when multiple copies of a program are running on different tasks.
Arguments	N/A
Example	**MOVE(length) AXIS(PROCNUMBER)**
See also	**PROC_STATUS**, **PROC**.

4-2-192 PSWITCH

Type	I/O command
Syntax	**PSWITCH(switch, enable [, axis, output_number, output_state, set_position, reset_position])**

Description

The **PSWITCH** command turns on an output when a predefined position is reached, and turns off the output when a second position is reached. The positions are specified as the measured absolute positions.

There are 16 position switches each of which can be assigned to any axis. Each switch has assigned its own ON and OFF positions and output number.

The command can be used with 2 or all 7 arguments. With only 2 arguments a given switch can be disabled.

PSWITCHes are calculated on each servo cycle and the output result applied to the hardware. The response time is therefore 1 servo cycle period approximately.

Note: An output may remain ON if it was ON when the **PSWITCH** was disabled. In such cases the **OP** command can be used to turn off an output as follows:

```
PSWITCH(2,OFF)
OP(14,OFF) ' Turn OFF pswitch controlling OP 14
```

Note: The physical switches that are used with **PSWITCH** are not fast hardware switches, so switching is done by software, which can introduce some small delays in operation. Fast hardware switching can be used only with an axis connected via the Encoder Interface port. Use the **HW_PSWITCH** command.

Arguments

- **switch**
  The switch number. Range: [0,15].
- **enable**
  The switch enable. Range: [on, off].
- **axis**
  The number of the axis providing the position input.
- **output_number**
  The physical output to set. Range: [8,31].
- **output_state**
  The state to output. Range: [on, off].
- **set_position**
  The absolute position in user units at which output is set.
- **reset_position**
  The absolute position in user units at which output is reset.

Example A rotating shaft has a cam operated switch which has to be changed for different size work pieces. There is also a proximity switch on the shaft to indicate the TDC of the machine. With a mechanical cam, the change from job to job is time consuming. This can be eased by using PSWITCH as a software cam switch. The proximity switch is wired to input 7 and the output is output 11. The shaft is controlled by axis 0. The motor has a 900ppr encoder. The output must be on from 80 units.
PSWITCH uses the unit conversion factor to allow the positions to be set in convenient units. First the unit conversion factor must be calculated and set. Each pulse on an encoder gives four edges for the CJ1W-MCH72 to count. There are thus 3,600 edges/rev or 10 edges/degree. If you set the unit conversion factor to 10, you can work in degrees.
Next you have to determine a value for all the **PSWITCH** arguments.
sw: The switch number can be any switch that is not in use. In this example, you will use number 0.
en: The switch must be enabled to work; set the enable to 1.
axis: The shaft is controlled by axis 0.
opno: The output being controlled is output 11.
opst: The output must be on so set to 1.
setpos: The output is to produced at 80 units.
rspos: The output is to be on for a period of 120 units.
This can all be put together in the following lines of BASIC code:

```
switch:
  UNITS AXIS(0) = 10 ' Set unit conversion factor
  REP_DIST = 360
  REP_OPTION = ON
  PSWITCH(0,ON,0,11,ON,80,200)
```

This program uses the repeat distance set to 360 degrees and the repeat option on so that the axis position will be maintained between 0 and 360 degrees.

See also **HW_PSWITCH**, **OP**, **UNITS**.

4-2-193 RAPIDSTOP

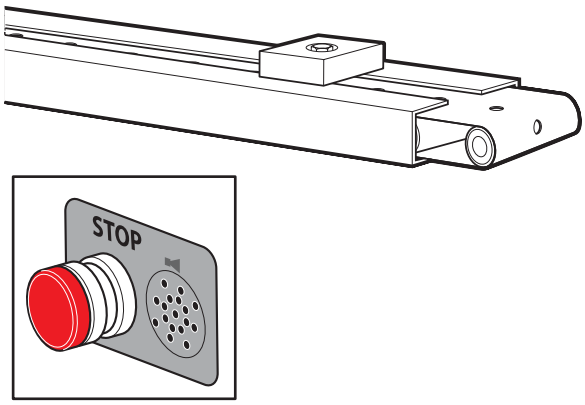
Type Axis command

Syntax **RAPIDSTOP**
RS

Description The **RAPIDSTOP** command cancels the current move on all axes from the current move buffer (**MTYPE**). Moves for speed profiled move commands (**MOVE**, **MOVEABS**, **MOVEMODIFY**, **FORWARD**, **REVERSE**, **MOVECIRC** and **MHELICAL**) will decelerate to a stop with the deceleration rate as set by the **DECEL** parameter. Moves for other commands will be immediately stopped.
Notes:

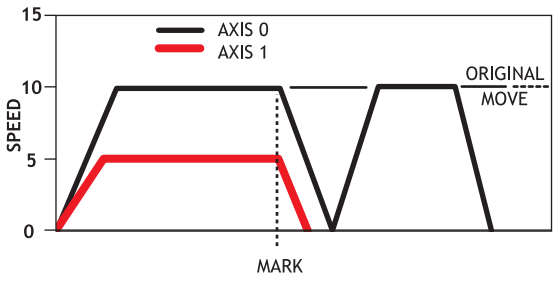
- **RAPIDSTOP** cancels only the presently executing moves. If further moves are buffered in the next move buffers (**NTYPE**) or the task buffers they will then be loaded.
- During the deceleration of the current moves additional **RAPIDSTOP**s will be ignored.

Arguments N/A



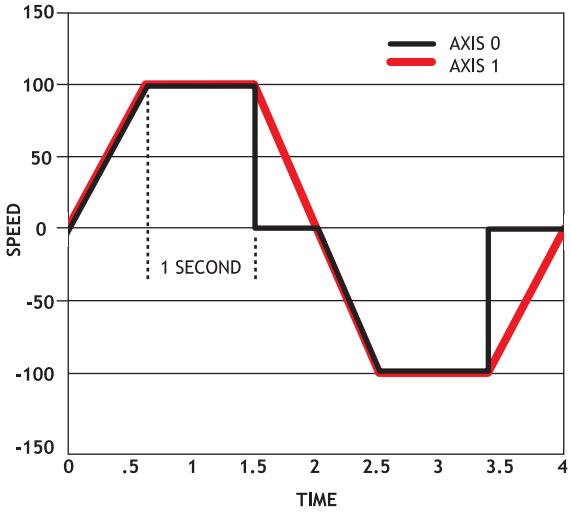
Example This example shows the implementation of a stop override button that cuts out all motion.

```
CONNECT (1,0) AXIS(1) 'axis 1 follows axis 0
BASE(0)
REPAEAT
MOVE(1000) AXIS (0)
MOVE(-100000) AXIS (0)
MOVE(100000) AXIS (0)
UNTIL IN (2)=OFF 'stop button pressed?
RAPIDSTOP
WA(10) 'wait to allow running move to cancel
RAPIDSTOP 'cancel the second buffered move
WA(10)
RAPIDSTOP 'cancel the third buffered move
```



Example This example shows the use of **RAPIDSTOP** to cancel a **MOVE** on the main axis and a **FORWARD** on the second axis. When the axes have stopped, a **MOVEABS** is applied to re-position the main axis.

```
BASE(0)
REGIST(3)
FORWARD AXIS(1)
MOVE (100000) 'apply a long move
WAIT UNTIL MARK
RAPIDSTOP
WAIT IDLE 'for MOVEABS to be accurate, the axis must stop
MOVEABS(3000)
```



Example This example shows the use of **RAPIDSTOP** to break a **CONNECT** and stop the motion. The connected axis stops immediately on the **RAPIDSTOP** command. The forward axis decelerates at the **DECEL** value.

```
BASE(0)
CONNECT(1,1)
FORWARD AXIS(1)
WAIT UNTIL VPSPEED=SPEED 'let the axis get to full speed
WA(1000)
RAPIDSTOP
WAIT IDLE AXIS(1) 'wait for axis 1 to decel
CONNECT(1,1) 're-connect axis 0
REVERSE AXIS(1)
WAIT UNTIL VPSPEED=SPEED
WA(1000)
RAPIDSTOP
WAIT IDLE AXIS(1)
```

See also **CANCEL**, **MTYPE**, **NTYPE**.

4-2-194 READ_BIT

Type System command

Syntax **READ_BIT(bit_number, vr_number)**

Description The **READ_BIT** command returns the value of the specified bit in the specified VR variable, either 0 or 1.

Arguments
- **bit_number**
  The number of the bit to be read. Range: [0,23].
- **vr_number**
  The number of the VR variable for which the bit is read. Range: [0,1023].

Example No example.

See also **CLEAR_BIT**, **SET_BIT**.

4-2-195 READ_OP

Type I/O command

Syntax **READ_OP(output_no)**
READ_OP(first_output_no, last_output_no)

Description **READ_OP(output_no)**, returns the binary value (0 or 1) of the digital output **output_no**.
READ_OP(first_output_no, last_output_no), returns the number that is the decimal representation of the concatenation of the binary values of the range **first_output_no** to **final_output_no**.
Note: The difference between **first_output_no** and **last_output_no** must be less than 24.
Note: Outputs 0 to 7 do not physically exist on the CJ1W-MCH72. They cannot be written. Their return value is always 0.
Note: **READ_OP** checks the state of the output logic. **READ_OP** can return the value 1 even if no actual output is present.

Arguments
- **output_no**
  The number of the output.
- **first_output_no**
  The number of the first output of the output range.
- **last_output_no**
  The number of the last output of the output range.

Example If output 11 has value 1, output 12 has value 1, output 13 has value 0, and output 14 has value 1, **READ_OP(11,14)** returns 13 (1101 bin).

Example In this example a single output is tested:
```
WAIT UNTIL READ_OP(12) = ON
GOSUB place
```

Example Check a range of 8 outputs and call a routine if one of them has value 1:
```
op_bits = READ_OP(16, 23)
IF op_bits <> 0 THEN
  GOSUB check_outputs
ENDIF
```

See also N/A

4-2-196 REG_POS

Type Axis parameter (read-only)

Syntax **REG_POS**

Description The **REG_POS** parameter stores the position in user units at which the primary registration event occurred.

Arguments N/A

Example A paper cutting machine uses a CAM profile shape to quickly draw paper through servo driven rollers, and stop the paper so it can be cut. The paper is printed with a registration mark. This mark is detected and the length of the next sheet is adjusted by scaling the CAM profile with the third parameter of the CAM command:

```
' Example Registration Program using CAM stretching:
' Set window open and close:
length=200
OPEN_WIN=10
CLOSE_WIN=length-10
GOSUB Initial
Loop:
  TICKS=0' Set millisecond counter to 0
  IF MARK THEN
    offset=REG_POS
    ' This next line makes offset -ve if at end of sheet:
    IF ABS(offset-length)<offset THEN offset=offset-length
    PRINT "Mark seen at:"offset[5.1]
  ELSE
    offset=0
    PRINT "Mark not seen"
  ENDIF
  ' Reset registration prior to each move:
  DEFPOS(0)
  REGIST(3+768)' Allow mark at first 10mm/last 10mm of sheet
  CAM(0,50,(length+offset*0.5)*cf,1000)
  WAIT UNTIL TICKS<-500
GOTO Loop
```

Note: variable **cf** is a constant that is calculated depending on the draw length of the machine per encoder edge.

See also **AXIS**, **MARK**, **REGIST**.

4-2-197 REG_POSB

Type Axis parameter (read-only)

Syntax **REG_POSB**

Description The **REG_POSB** parameter stores the position in user units at which the secondary registration event occurred.

Arguments N/A

Example No example.

See also **AXIS**, **MARKB**, **REGIST**.

4-2-198 REGIST

Type Axis command

Syntax **REGIST(mode)**

Description

The **REGIST** command sets up the registration operation. The command captures an axis position when a registration signal is detected. With an Encoder Interface the capture is done by the hardware, so software delays do not affect the accuracy of the position that is captured. With a MECHATROLINK-II axis, the capture is done by the Servo Driver.

With an Encoder Interface, a **REGIST** command can capture two registration positions using separate registration inputs. When a primary registration event has occurred, the **MARK** axis parameter is set to ON and the position is stored in the **REG_POS** axis parameter. For the secondary registration event, the **MARKB** axis parameter is set to ON and the position is stored in the **REG_POSB** axis parameter. MARKB and REG_POSB are applicable only to axes with **ATYPE** values 43, 44 and 45.

MECHATROLINK-II registration can be performed using encoder Z-marker or external registration inputs EXT1, EXT2 or EXT3 on a Servo Driver. Unlike Flexible axis axes, only one registration position can be captured. When a registration event has occurred, the **MARK** axis parameter is set to ON and the position is stored in the **REG_POS** axis parameter.

The registration signals EXT1, EXT2 and EXT3 must be allocated to CN1 inputs with the driver parameter Pn511. For example Pn511=654x sets the connections of EXT1 to CN1 pin44, EXT2 to CN1 pin45 and EXT3 to CN1 pin46 of the Sigma II Servo Driver.

The table below shows how to configure the external inputs individually.

Note: To configure EXT1, EXT2 and EXT3 registration signals parameter numbers Pn511.1, Pn511.2 and Pn511.3 are used respectively. Pn511.0 is not used. Refer to the user manual of the Servo Driver for more details.

Registration signal	Parameter number	Parameter value	Description
EXT 1	Pn511.1	0 to 3	Not used
		4	Input from CN1 pin44 (Rising edge)
		5	Input from CN1 pin45 (Rising edge).
		6	Input from CN1 pin46 (Rising edge).
		7	Signal always OFF.
		8	Signal always ON.
		9 to C	Not used
		D	Input from CN1 pin44 (Falling edge).
		E	Input from CN1 pin45 (Falling edge).
		F	Input from CN1 pin46 (Falling edge).
EXT 2	Pn511.2	As for EXT 1	As for EXT 1
EXT 3	Pn511.3	As for EXT 1	As for EXT 1

Note The mapping of the registration signals in the table above applies to the Sigma-II Servo Driver. For the mapping of the registration signals of the Sigma-V Servo Driver, refer to section 5-1-6.

Inclusive windowing lets the registration to occur only within a specified window of axis positions. With this windowing function, registration events are ignored if the axis measured position is not greater than the **OPEN_WIN** axis parameter, and less than the **CLOSE_WIN** parameter. Exclusive windowing allows the registration to occur only outside of the specified window of axis positions. With this windowing function, the registration events are ignored if the axis measured position is not less than the **OPEN_WIN** axis parameter, and greater than the **CLOSE_WIN** parameter.

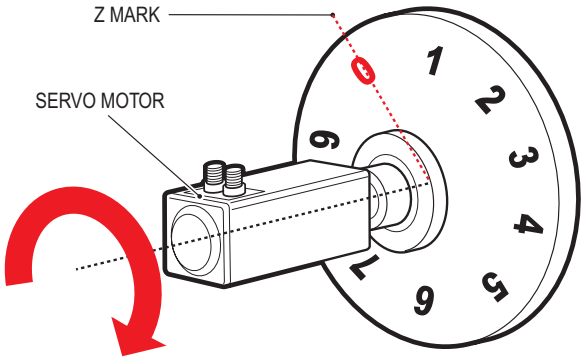
Arguments

- **mode**
  The mode parameter specifies the registration input and event for use and the signal edge the registration event occurs. The mode parameter also specifies the use of the windowing function and filtering.
  The mode parameter differs between MECHATROLINK-II and Encoder Interface. The function of each bit in the mode parameter is explained in the tables below.

Bit	Function (MECHATROLINK-II)
1, 0	Primary registration occurs for: • 00: Z-mark of the encoder • 01: EXT1 input (CN1 pin programmed with Pn511.1) • 10: EXT2 input (CN1 pin programmed with Pn511.2) • 11: EXT3 input (CN1 pin programmed with Pn511.3)
2 -7	Not used
9, 8	Windowing function choice: • 00: No windowing • 01: Inclusive windowing • 10: Inclusive windowing • 11: Exclusive windowing
10	Not used

Bit	Function (Encoder Interface)
1, 0	Primary registration occurs for: • 00: Z-mark of the encoder • 01: REG 0 input • 10: REG 1 input
2	Set this bit to use primary registration event
3	Primary registration event occurs on signal: • 0: rising edge • 1: falling edge
5, 4	Secondary registration occurs for: • 00: Z-mark of the encoder • 01: REG 0 input • 10: REG 1 input
6	Set this bit to use secondary registration event
7	Secondary registration event occurs on signal: • 0: rising edge • 1: falling edge

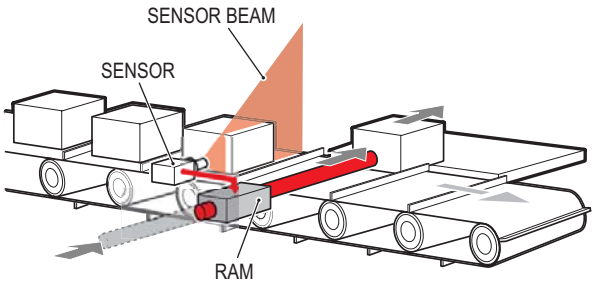
Bit	Function (Encoder Interface)
9, 8	Windowing function choice: • 00: No windowing • 01: Inclusive windowing • 10: Inclusive windowing • 11: Exclusive windowing
10	Set this bit to use filtering function



Example

A disc used in a laser printing process requires registration to the Z marker before it can start to print. The example code locates to the Z marker, and then sets it as the zero position.

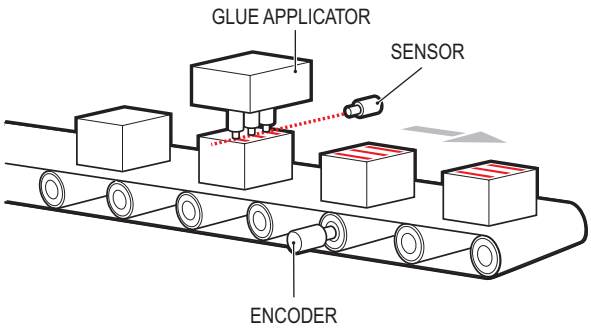
```
REGIST(1) 'set registration point on Z mark
FORWARD 'start movement
WAIT UNTIL MARK
CANCEL 'stops movement after Z mark
WAIT IDLE
MOVEABS (REG_POS) 'relocate to Z mark
WAIT IDLE
DEFPOS(0) 'set zero position
```



Example

Components are placed on a flighted belt. The flights are 120 mm apart. The components are on the belt 30 mm from the flights. When a component is found, an actuator pushes it off the belt. To prevent that the sensor finds a flight instead of a component, registration with windowing is used.

```
REP_DIST=120 ‘sets repeat distance to pitch of belt flights
REP_OPTION=ON
OPEN_WIN=30 ' sets window open position
CLOSE_WIN=90 ' sets window close position
REGIST(4+256) ' R input registration with windowing
FORWARD ' start the belt
box_seen=0
REPEAT
  WAIT UNTIL MPOS<60 ' wait for centre point between flights
  WAIT UNTIL MPOS>60 ' so that actuator is fired between flights
  IF box_seen=1 THEN ' was a box seen on the previous cycle?
    OP(8,ON) ' fire actuator
    WA(100)
    OP(8,OFF) ' retract actuator
    box_seen=0
  ENDIF
  IF MARK THEN box_seen=1 ' set “box seen” flag
  REGIST(4+256)
UNTIL IN(2)=OFF
CANCEL ' stop the belt
WAIT IDLE
```



Example A machine adds glue to the top of a box. To do this, it must switch output 8. It must detect the rising edge (appearance) and the falling edge (end) of a box. Also, the MPOS must be set to zero when the Z position is detected.

```
reg=6 'select registration mode 6 (rising edge R, rising edge Z)
REGIST(reg)
FORWARD
WHILE IN(2)=OFF
 IF MARKB THEN 'on a Z mark mpos is reset to zero
  OFFPOS=-REG_POSB
  REGIST(reg)
 ELSEIF MARK THEN 'on R input output 8 is toggled
  IF reg=6 THEN
   'select registration mode 8 (falling edge R, rising edge Z)
   reg=8
   OP(8,ON)
  ELSE
   reg=6
   OP(8,OFF)
  ENDIF
  REGIST(reg)
 ENDIF
WEND
CANCEL
```

See also **AXIS**, **MARK**, **MARKB**, **REG_POS**, **REG_POSB**, **OPEN_WIN**, **CLOSE_WIN**.

Note The current Trajexia firmware version (1.6652) does not properly support this command for Sigma-V Servo Drivers.

4-2-199 REMAIN

Type Axis parameter (read-only)

Syntax **REMAIN**

Description The **REMAIN** parameter contains the distance remaining to the end of the current move. It can be checked to see how much of the move has been completed.

The units in which **REMAIN** is expressed depends on the type of the motion command:

- If a master axis is moved by **MOVELINK** or **CAMBOX**, **REMAIN** is expressed in user units set by **UNITS**.
- If a slave axis is moved by **MOVELINK** or **CAMBOX**, **REMAIN** is expressed in encoder counts.
- If a master or a slave axis is moved by a motion command that is not **MOVELINK** or **CAMBOX**, **REMAIN** is expressed in user units set by **UNITS**.

The **CONNECT** command moves an axis without a defined end. For this command, **REMAIN** has the fixed value of 1000.

Arguments N/A

Example To change the speed to a slower value 5mm from the end of a move.

```
start:
  SPEED = 10
  MOVE(45)
  WAIT UNTIL REMAIN < 5
  SPEED = 1
  WAIT IDLE
```

See also **AXIS**, **UNITS**

4-2-200 REMOTE_ERROR

Type Axis parameter

Syntax **REMOTE_ERROR**

Description Returns the number of errors on the MECHATROLINK-II communication link of a driver.

Arguments N/A

Example

```
>>PRINT REMOTE_ERROR
1.0000
```

See also N/A

4-2-201 RENAME

Type Program command

Syntax **RENAME "old_program_name" "new_program_name"**

Description The **RENAME** command changes the name of a program in the CJ1W-MCH72 directory. The program names can also be specified without quotes.
Note: This command is implemented for a Command Line Terminal only and should not be used from within programs.

Arguments
- **old_program_name**
  The current name of the program.
- **new_program_name**
  The new name of the program.

Example **RENAME "car" "voiture"**

See also **COPY**, **DEL**, **NEW**.

4-2-202 REP_DIST

Type Axis parameter

Syntax **REP_DIST**

Description The **REP_DIST** parameter contains the repeat distance, which is the allowable range of movement for an axis before the demand position (**DPOS**) and measured position (**MPOS**) are corrected. **REP_DIST** is defined in user units. The exact range is controlled by **REP_OPTION**. The **REP_DIST** can have any non-0 positive value.
When the measured position has reached its limit, the CJ1W-MCH72 will adjust the absolute positions without affecting the move in progress or the servo algorithm. Note that the demand position can be outside the range because the measured position is used to trigger the adjustment. When measured position reaches REP_DIST, twice that distance is subtracted to ensure that the axis always stays in the range [-REP_DIST, REP_DIST], assuming that REP_OPTION=OFF, or in the range [0, REP_OPTION], assuming that REP_OPTION=ON.
For every occurrence (**DEFPOS**, **OFFPOS**, **MOVEABS**, **MOVEMODIFY**) which defines a position outside the range, the end position will be redefined within the range.
The default value for all axes is 5000000.

Arguments N/A

Example No example.

See also **AXIS**, **DPOS**, **MPOS**, **REP_OPTION**, **UNITS**.

4-2-203 REP_OPTION

Type Axis parameter

Syntax **REP_OPTION**

Description The **REP_OPTION** parameter controls the application of the **REP_DIST** axis parameter and the repeat option of the **CAMBOX** and **MOVELINK** Axis commands. The default value is 0. See the table below.

Bit	Description
0	The repeated distance range is controlled by bit 0 of the **REP_OPTION** parameter. • If **REP_OPTION** bit 0 is off, the range of the demanded and measured positions will be between **-REP_DIST** and **REP_DIST**. • If **REP_OPTION** bit 0 is on, the range of the demanded and measured positions will be between 0 and **REP_DIST**.
1	The automatic repeat option of the **CAMBOX** and **MOVELINK** commands are controlled by bit 1 of the **REP_OPTION** parameter. The bit is set on to request the system software to end the automatic repeat option. When the system software has set the option off it automatically clears bit 1 of **REP_OPTION**.

Arguments N/A

Example No example.

See also **AXIS**, **CAMBOX**, **MOVELINK**, **REP_DIST**.

4-2-204 REPEAT..UNTIL

Type Program control command

Syntax
REPEAT
commands
UNTIL condition

Description
The **REPEAT ... UNTIL** structure allows the program segment between the **REPEAT** and the **UNTIL** statement to be repeated a number of times until the condition becomes **TRUE**.
Note: **REPEAT ... UNTIL** construct can be nested indefinitely.

Arguments
- **commands**
  Any valid set of BASIC commands
- **condition**
  Any valid BASIC logical expression

Example
A conveyor is to index 100mm at a speed of 1000mm/s, wait for 0.5s and then repeat the cycle until an external counter signals to stop by turning on input 4.
cycle:

```
SPEED = 1000
REPEAT
  MOVE(100)
  WAIT IDLE
  WA(500)
UNTIL IN(4) = ON
```

See also
FOR..TO..STEP..NEXT, **WHILE..WEND**.

4-2-205 RESET

Type
System command

Syntax
RESET

Description
The **RESET** command sets the value of all local variables of the current BASIC task to 0.

Arguments
N/A

Example
No example.

See also
CLEAR.

4-2-206 RETURN

See **GOSUB..RETURN**.

4-2-207 REV_IN

Type
Axis parameter

Syntax
REV_IN

Description The **REV_IN** parameter contains the input number to be used as a reverse limit input. The valid input range is 0 to 31. Values 0 to 15 represent physically present inputs of CJ1W-MCH72 I/O connector and are common for all axes.
Values 16 to 31 are mapped directly to driver inputs that are present on the CN1 connector. They are unique for each axis. It depends on the type of Servo Driver which Servo Driver inputs are mapped into inputs 16 to 31. For more information on Servo Driver I/O mapping into the Trajexia I/O space, refer to section 5-1-4.
As default the parameter is set to -1, no input is selected.
If an input number is set and the limit is reached, any reverse motion on that axis will be stopped. Bit 5 of the **AXISSTATUS** axis parameter will also be set.
Note: This input is active low.

Arguments N/A

Example No example.

See also **AXIS**, **AXISSTATUS**, **FWD_IN**.

4-2-208 REV_JOG

Type Axis parameter

Syntax **REV_JOG**

Description The **REV_JOG** parameter contains the input number to be used as a jog reverse input. The input can be from 0 to 31. As default the parameter is set to -1, no input is selected.
Note: This input is active low.

Arguments N/A

Example No example.

See also **AXIS**, **FAST_JOG**, **FWD_JOG**, **JOGSPEED**, **UNITS**.

4-2-209 REVERSE

Type Axis command

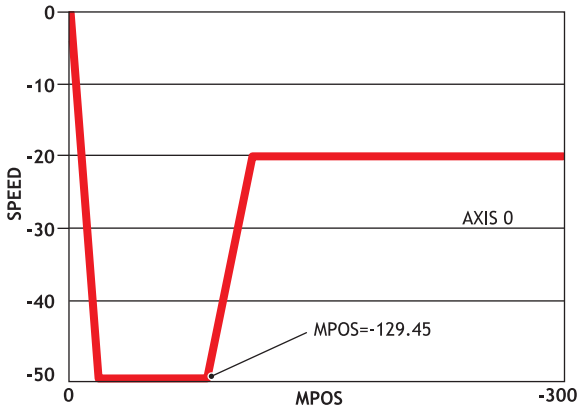
Syntax **REVERSE**
RE

Description The **REVERSE** command moves an axis continuously in reverse at the speed set in the **SPEED** parameter. The acceleration rate is defined by the **ACCEL** axis parameter.
REVERSE works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note: The reverse motion can be stopped by executing the **CANCEL** or **RAPIDSTOP** command, or by reaching the reverse limit, inhibit, or origin return limit.

Arguments N/A

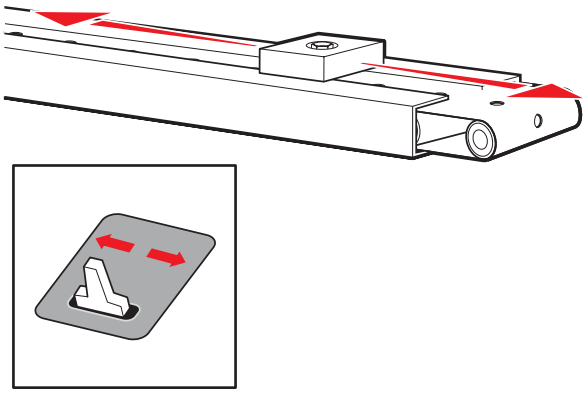
Example Run an axis in reverse. When an input signal is detected on input 5, stop the axis.

```
back:
  REVERSE
  WAIT UNTIL IN(0) = ON ' Wait for stop signal
  CANCEL
```



Example Run an axis in reverse. When it reaches a certain position, slow down.

```
DEFPOS(0) ' set starting position to zero
REVERSE
WAIT UNTIL MPOS<-129.45
SPEED=slow_speed
WAIT UNTIL VP_SPEED=slow_speed ' wait until the axis slows
OP(11,ON) ' turn on an output to show that speed is now slow
```



Example A joystick is used to control the speed of a platform. A deadband is required to prevent oscillations from the joystick midpoint. This is done with the **REVERSE** command, which sets the correct direction relative to the operator. Then, the joystick adjusts the speed through analogue input 0.

```
REVERSE
WHILE IN(2)=ON
 IF AIN(0)<50 AND AIN(0)>-50 THEN 'sets a deadband in the input
  SPEED=0
 ELSE
  SPEED=AIN(0)*100 'sets speed to a scale of AIN
 ENDIF
WEND
CANCEL
```

See also **AXIS**, **CANCEL**, **FORWARD**, **RAPIDSTOP**.

4-2-210 RS_LIMIT

Type Axis parameter

Syntax **RS_LIMIT**
RSLIMIT

Description The **RS_LIMIT** parameter contains the absolute position of the reverse software limit in user units.
A software limit for reverse movement can be set from the program to control the working range of the machine. When the limit is reached, the CJ1W-MCH72 will decelerate to 0, and then cancel the move. Bit 10 of the **AXISSTATUS** axis parameter will be turned on while the axis position is smaller than / below **RS_LIMIT**.

Arguments N/A

Example No example.

See also **AXIS**, **FS_LIMIT**, **UNITS**.

4-2-211 RUN

Type Program command

Syntax **RUN ["program_name" [, task_number]]**

Description The **RUN** command executes the program in the CJ1W-MCH72 as specified with **program_name**. **RUN** with the program name specification will run the current selected program. The program name can also be specified without quotes.
The task number specifies the task number on which the program will be run. If the task number is omitted, the program will run on the highest available task. **RUN** can be included in a program to run another program.
Note: Execution continues until one of the following occurs:

- There are no more lines to execute.
- **HALT** is typed at the command line to stop all programs.
- **STOP** is typed at the command line to stop a single program.
- The **STOP** command in the program is encountered.
- A run-time error is encountered.

Arguments
- **program_name**
  Any valid program name.
- **task_number**
  Any valid task number. Range: [1,14].

Example
>> SELECT "PROGRAM"
PROGRAM selected
>> RUN
This example executes the currently selected program.

Example
RUN "sausage"
This example executes the program named **sausage**.

Example
RUN "sausage",3
This example executes the program named **sausage** on task 3.

See also
HALT, **STOP**.

4-2-212 RUN_ERROR

Type
Task parameter (read-only)

Syntax
RUN_ERROR

Description
The **RUN_ERROR** parameter contains the number of the last BASIC run-time error that occurred on the specified task.
Each task has its own **RUN_ERROR** parameter. Use the **PROC** modifier to access the parameter for a certain task. Without **PROC** the current task will be assumed.
The table below gives an overview of error numbers and the associated error messages.

Number	Message	Number	Message
1	Command not recognized	70	Value is incorrect
2	Invalid transfer type	71	Invalid I/O channel
3	Error programming Flash	72	Value cannot be set. Use **CLEAR_PARAMS** command
4	Operand expected	73	Directory not locked
5	Assignment expected	74	Directory already locked
6	QUOTES expected	75	Program not running on this process
7	Stack overflow	76	Program not running
8	Too many variables	77	Program not paused on this process
9	Divide by zero	78	Program not paused
10	Extra characters at end of line	79	Command not allowed when running Trajexia Studio
11	] expected in **PRINT**	80	Directory structure invalid
12	Cannot modify a special program	81	Directory is locked
13	**THEN** expected in **IF**/**ELSEIF**	82	Cannot edit program

Number	Message	Number	Message
14	Error erasing Flash	83	Too many nested OPERANDS
15	Start of expression expected	84	Cannot reset when drive servo on
16	) expected	85	Flash Stick blank
17	, expected	86	Flash Stick not available on this controller
18	Command line broken by ESC	87	Slave error
19	Parameter out of range	88	Master error
20	No process available	89	Network timeout
21	Value is read only	90	Network protocol error
22	Modifier not allowed	91	Global definition is different
23	Remote axis is in use	92	Invalid program name
24	Command is command line only	93	Program corrupt
25	Command is runtime only	94	More than one program running when trying to set GLOBAL/CONSTANT
26	LABEL expected	95	Program encrypted
27	Program not found	96	TOKEN definition incorrect
28	Duplicate label	97	Cannot change program type once it has been created
29	Program is locked	98	Command expected
30	Program(s) running	99	Invalid command
31	Program is stopped	100	Invalid parameter for command
32	Cannot select program	101	Too many tokens in block
33	No program selected	102	Invalid mix of modal groups
34	No more programs available	103	Variable defined outside include file
35	Out of memory	104	Invalid program type
36	No code available to run	105	Variable not declared
37	Command out of context	106	(expected
38	Too many nested structures	107	Number expected
39	Structure nesting error	108	AS expected
40	**ELSE**/**ELSEIF**/**ENDIF** without previous **IF**	109	STRING, VECTOR or ARRAY expected
41	**WEND** without previous **WHILE**	110	String expected
42	**UNTIL** without previous **REPEAT**	111	Invalid **MSPHERICAL** input
43	Variable expected	112	Too many labels

Number	Message	Number	Message
44	**TO** expected after **FOR**	113	Symbol table locked
45	Too may nested **FOR**/**NEXT**	114	Incorrect symbol type
46	**NEXT** without **FOR**	115	Invalid mix of data types
47	**UNTIL**/**IDLE** expected after **WAIT**	116	Command not allowed when running Trajexia Studio
48	**GOTO**/**GOSUB** expected	117	Parameter expected
49	Too many nested **GOSUB**	118	Firmware error: Device in use
50	**RETURN** without **GOSUB**	119	Device error: Timeout waiting for device
51	LABEL must be at start of line	120	Device error: Command not supported by device
52	Cannot nest one line **IF**	121	Device error: CRC error
53	LABEL not found	122	Device error: Error writing to device
54	LINE NUMBER cannot have decimal point	123	Device error: Invalid response from device
55	Cannot have multiple instances of REMOTE	124	Firmware error: Cannot reference data outside current block
56	Invalid use of $	125	Disk error: Invalid MBR
57	**VR(x)** expected	126	Disk error: Invalid boot sector
58	Program already exists	127	Disk error: Invalid sector/cluster reference
59	Process already selected	128	File error: Disk full
60	Duplicate axes not permitted	129	File error: File not found
61	PLC type is invalid	130	File error: Filename already exists
62	Evaluation error	131	File error: Invalid filename
63	Reserved keyword not available on this controller	132	File error: Directory full
64	VARIABLE not found	133	Command only allowed when running Trajexia Studio
65	Table index range error	134	# expected
66	Features enabled do not allow **ATYPE** change	135	**FOR** expected
67	Invalid line number	136	INPUT/OUTPUT/APPEND expected
68	String exceeds permitted length	137	File not open
69	Scope period should exceed number of **AIN** parameters	138	End of file

Arguments N/A

Example **>> PRINT RUN_ERROR PROC(5)**
9.0000

See also **BASICERROR**, **ERROR_LINE**, **PROC**.

4-2-213 RUNTYPE

Type Program command

Syntax **RUNTYPE "program_name", auto_run [, task_number]**

Description The **RUNTYPE** command determines whether the program, specified by **program_name**, is run automatically at start-up or not and which task it is to run on. The task number is optional, if omitted the program will run at the highest available task.
The current **RUNTYPE** status of each programs is displayed when a **DIR** command is executed. If any program has compilation errors no programs will be started at power up. To set the **RUNTYPE** using Trajexia Studio, set the **Priority** property of the program.

Arguments
- **program_name**
  The name of the program whose **RUNTYPE** is being set.
- **auto_run**
  0 = Running manually on command; 1 = Automatically execute on power up. All non-zero values are considered as 1.
- **task_number**
  The number of the task on which to execute the program. Range: [1, 14].

Example **>> RUNTYPE progname,1,3**
This line sets the program **progname** to run automatically at start-up on task 3.

Example **>> RUNTYPE progname,0**
This line sets the program **progname** to manual running.

See also **AUTORUN**, **EPROM**, **EX**.

4-2-214 S_REF

Type Axis parameter

Syntax **S_REF**

Description This parameter contains the speed reference value which is applied directly to the Servo Driver when the axis is in open loop (**SERVO=OFF**). The range of this parameter is defined by the number of available bits. For MECHATROLINK-II axes, **S_REF** takes 32 bits, so the available range is [-2147483648, 2147483648], which corresponds to a voltage range [-10V, 10V]. For Flexible axis axes, **S_REF** takes 16 bits, so the available range is [-32768, 32767], which corresponds to a voltage range [-10V, 10V]. These ranges can be limited by using the **OUTLIMIT** parameter.
The value currently being applied to the driver can be read using the **S_REF_OUT** axis parameter.

Arguments N/A

Example
```
WDOG = ON
SERVO = OFF
square:
   S_REF AXIS(0) = 2000
   WA(250)
   S_REF AXIS(0) = -2000
   WA(250)
GOTO square
```
These lines can be used to force a square wave of positive and negative movement with a period of approximately 500ms on axis 0.

See also **AXIS**, **S_REF_OUT**, **OUTLIMIT**, **SERVO**.

4-2-215 S_REF_OUT

Type Axis parameter (read-only)

Syntax **S_REF_OUT**

Description The **S_REF_OUT** parameter contains the speed reference value being applied to the Servo Driver for both open and closed loop.
In closed loop (**SERVO=ON**), the motion control algorithm will output a speed reference signal determined by the control gain settings and the Following Error. The position of the servo motor is determined using the Axis commands. In open loop (**SERVO=OFF**), the speed reference signal is determined by the **S_REF** axis parameter.

Arguments N/A

Example
```
>> PRINT S_REF_OUT AXIS(0)
288.0000
```

See also **AXIS**, **S_REF**, **OUTLIMIT**, **SERVO**.

4-2-216 SCOPE

Type System command

Syntax **SCOPE(control, period, table_start, table_stop, P0 [, P1 [, P2 [, P3]]])**

Description The **SCOPE** command programs the system to automatically store up to 4 parameters every sample period. The storing of data will start as soon as the **TRIGGER** command has been executed.
The sample period can be any multiple of the servo period. The parameters are stored in the TABLE array and can then be read back to a computer and displayed on the Trajexia Studio data trace or written to a file for further analysis using the **Create Table file** option on the **File** menu.
The current TABLE position for the first parameter which is written by **SCOPE** can be read from the **SCOPE_POS** parameter.
Notes:
1.Trajexia Studio uses the **SCOPE** command when running the Oscilloscope function.
2.To minimize calculation time for writing the real-time data, the **SCOPE** command is writing raw data to the TABLE array. For example
a)The parameters are written in encoder edges (per second) and therefore not compensated for the **UNITS** conversion factor.
b)The **MSPEED** parameter is written as the change in encoder edges per servo period.
3.Applications like the **CAM** command, **CAMBOX** command and the **SCOPE** command all use the same TABLE as the data area.

Arguments
- **control**
  Set on or off to control **SCOPE** execution. If turned on the **SCOPE** is ready to run as soon as the **TRIGGER** command is executed.
- **period**
  The number of servo periods between data samples.
- **table_start**
  The address of the first element in the TABLE array to start storing data.
- **table_stop**
  The address of the last element in the TABLE array to be used.
- **P0**
  First parameter to store.
- **P1**
  Optional second parameter to store.
- **P2**
  Optional third parameter to store.
- **P3**
  Optional fourth parameter to store.

Example **SCOPE(ON,10,0,1000,MPOS AXIS(1),DPOS AXIS(1))**
This example programs the **SCOPE** function to store the **MPOS** parameter for axis 1 and the **DPOS** parameter for axis 1 every 10 servo cycles. The **MPOS** parameter will be stored in TABLE locations 0 to 499; the **DPOS** parameters, in TABLE locations 500 to 999. The **SCOPE** function will wrap and start storing at the beginning again unless stopped. Sampling will not start until the **TRIGGER** command is executed.

Example **SCOPE(OFF)**
This above line turns the scope function off.

See also **SCOPE_POS**, **TABLE**, **TRIGGER**.

4-2-217 SCOPE_POS

Type	System parameter (read-only)
Syntax	**SCOPE_POS**
Description	The **SCOPE_POS** parameter contains the current TABLE position at which the **SCOPE** command is currently storing its parameters.
Arguments	N/A
Example	No example.
See also	**SCOPE**.

4-2-218 SELECT

Type	Program command
Syntax	**SELECT "program_name"**
Description	The **SELECT** command specifies the current program for editing, running, listing, etc. **SELECT** makes a new program if a program with the name entered does not exist. The program name can also be specified without quotes. When a program is selected, the commands **COMPILE**, **DEL**, **EDIT**, **LIST**, **NEW**, **RUN**, **STEPLINE**, **STOP** and **TROFF** will apply to the currently selected program unless a program name is specified in the command line. When another program is selected, the previously selected program will be compiled. The selected program cannot be changed when a program is running. Note: This command is implemented for a Command Line Terminal. Trajexia Studio automatically selects programs when you click on their entry in the list in the control panel.
Arguments	N/A
Example	**>> SELECT "PROGRAM"** **PROGRAM selected** **>> RUN**
See also	**COMPILE**, **DEL**, **EDIT**, **LIST**, **NEW**, **RUN**, **STEPLINE**, **STOP**, **TROFF**.

4-2-219 SERVO

Type	Axis parameter
Syntax	**SERVO**
Description	The **SERVO** parameter determines whether the base axis runs under servo closed loop control (**SERVO=ON**) or open loop (**SERVO=OFF**). In closed loop, the motion control algorithm will output a speed reference signal determined by the control gain settings and the Following Error. The position of the servo motor is determined using the Axis commands. In open loop, the speed reference signal is completely determined by the **S_REF** axis parameter.
Arguments	N/A
Example	**SERVO AXIS(0) = ON ' Axis 0 is under servo control** **SERVO AXIS(1) = OFF ' Axis 1 is run open loop**

See also **AXIS**, **FE_LIMIT**, **S_REF**, **S_REF_OUT**, **WDOG**.

4-2-220 SERVO_PERIOD

Type System parameter

Syntax **SERVO_PERIOD**

Description The **SERVO_PERIOD** parameter sets the servo cycle period of the CJ1W-MCH72. The timing of the execution of the program tasks and the refreshing of the control data and I/O of the Unit are all depending on this setting. The parameter is defined in microseconds. The CJ1W-MCH72 can be set in either 0.5, 1.0, 2.0 or 4.0 ms servo cycle. See the table below. The controller must be reset before the new servo period will be applied.

Value	Description
500	0.5 ms
1000	1.0 ms
2000	2.0 ms
4000	4.0 ms

Arguments N/A

Example No example.

See also EX.

⚠ **Caution** When the parameter has been set, a power down or software reset (using **EX**) must be performed for the complete system. Not doing so may result in undefined behaviour.

4-2-221 SET_BIT

Type System command

Syntax **SET_BIT(bit_number, vr_number)**

Description The **SET_BIT** command sets the specified bit in the specified VR variable to one. Other bits in the variable will keep their values.

Arguments
- **bit_number**
  The number of the bit to be set. Range: [0,23].
- **vr_number**
  The number of the VR variable for which the bit is set. Range: [0,1023].

Example No example.

See also **CLEAR_BIT**, **READ_BIT**, **VR**.

4-2-222 SGN

Type Mathematical function

Syntax **SGN(expression)**

Description The **SGN** function returns the sign of a number. It returns value 1 for positive values (including 0) and value -1 for negative values.

Arguments
- **expression**
  Any valid BASIC expression.

Example **>> PRINT SGN(-1.2)**
-1.0000

See also N/A

4-2-223 SIN

Type Mathematical function

Syntax **SIN(expression)**

Description The **SIN** function returns the sine of the expression. Input values are in radians and may have any value. The result value will be in the range from -1 to 1.

Arguments
- **expression**
  Any valid BASIC expression.

Example **>> PRINT SIN(PI/2)**
1.0000

See also N/A

4-2-224 SPEED

Type Axis parameter

Syntax **SPEED**

Description The **SPEED** parameter contains the demand speed for an axis in units/s. It can have any positive value (including 0). The demand speed is the maximum speed for the speed profiled motion commands.

Arguments N/A

Example
```
SPEED = 1000
PRINT "Set speed = ";SPEED
```

See also **ACCEL**, **AXIS**, **DATUM**, **DECEL**, **FORWARD**, **MOVE**, **MOVEABS**, **MOVECIRC**, **MOVEMODIFY**, **REVERSE**, **UNITS**.

4-2-225 SPEED_SIGN

Type Axis parameter

Syntax **SPEED_SIGN**

Description The **SPEED_SIGN** parameter configures the voltage range of the analogue speed reference output of the Encoder Interface when the axis type **ATYPE** is set to 44.
If **SPEED_SIGN = OFF**, the voltage range of the analogue speed reference output is [-10V, 10V]. The positive reference voltage corresponds to forward movements, in which case **DPOS** and **MPOS** increment. The negative reference voltage corresponds to reverse movements, in which case **DPOS** and **MPOS** decrement. OFF is the default setting at power-on.
If **SPEED_SIGN = ON**, the voltage range of the analogue speed reference output is [0V, 10V]. The OUT1 signal of the Encoder Interface for the corresponding axis is used as a direction signal. During forward movements, the controller sets OUT1 to OFF. During reverse movements, the controller sets OUT1 to ON. This setting is to be used for Servo Drivers that require both speed and direction signals as a speed reference.

Arguments N/A

Example No example.

See also **ATYPE**, **S_REF**, **S_REF_OUT**.

4-2-226 SQR

Type Mathematical function

Syntax **SQR(expression)**

Description The **SQR** function returns the square root of the expression. The expression must have positive (including 0) value.

Arguments
- **expression**
  Any valid BASIC expression.

Example
```
>> PRINT SQR(4)
2.0000
```

See also N/A

4-2-227 SRAMP

Type Axis parameter

Syntax **SRAMP**

Description The **SRAMP** parameter contains the S-curve factor. The S-curve factor controls the amount of rounding applied to the trapezoidal profiles. A value of 0 sets no rounding. A value of 10 sets maximum rounding. The default value of the parameter is 0.
SRAMP is applied to the **FORWARD**, **MOVE**, **MOVEABS**, **MOVECIRC**, **MHELICAL** and **REVERSE** commands.
Notes:
- Using S-curves increases the time required for the movement to complete.
- The S-curve factor must not be changed while a move is in progress.

Arguments N/A

Example No example.

See also **AXIS**.

4-2-228 STEP

See **FOR..TO..STEP..NEXT**.

4-2-229 STEP_RATIO

Type Axis command

Syntax **STEP_RATIO(output_count, dpos_count)**

Description This command sets up a ratio for the output of the stepper axis. Every servo-period the number of steps is passed through the **STEP_RATIO** function before it goes to the step pulse output.
Pulse Count Out = (numerator)/(denominator) * **MPOS**.
STEP_RATIO affects both **MOVECIRC** and **CAMBOX**.
Notes:

- The **STEP_RATIO** function operates before the divide by 16 factor in the stepper axis.
- Large ratios should be avoided as they will lead to either loss of resolution or much reduced smoothness in the motion. The actual physical step size x 16 is the BASIC resolution of the axis and use of this command may reduce the ability of the Motion Controller to accurately achieve all positions.
- **STEP_RATIO** does not replace **UNITS**. Do not use **STEP_RATIO** to remove the x16 factor on the stepper axis as this will lead to poor step frequency control.

Arguments

- **denominator**
  An integer number between 0 and 16777215 that is used to define the denominator in the above equation.
- **numerator**
  An integer number between 0 and 16777215 that is used to define the numerator in the above equation.

Example Two axes are set up as X and Y but the axes ' steps per mm are not the same. Interpolated moves require identical UNITS values on both axes in order to keep the path speed constant and for MOVECIRC to work correctly. The axis with the lower resolution is changed to match the higher step resolution axis so as to maintain the best accuracy for both axes.

```
' Axis 0: 500 counts per mm (31.25 steps per mm)
' Axis 1: 800 counts per mm (50.00 steps per mm)
BASE(0)
STEP_RATIO(500,800)
UNITS = 800
BASE(1)
UNITS = 800
```

See also N/A

4-2-230 STEPLINE

Type Program command

Syntax **STEPLINE ["program_name" [, task_number]]**

Description The **STEPLINE** command executes one line (i.e., "steps") in the program specified by **program_name**. The program name can also be specified without quotes. If **STEPLINE** is executed without program name on the command line the current selected program will be stepped. If **STEPLINE** is executed without program name in a program this program will be stepped.
If the program is specified then all occurrences of this program will be stepped. A new task will be started when there is no copy of the program running. If the task is specified as well then only the copy of the program running on the specified task will be stepped. If there is no copy of the program running on the specified task then one will be started on it.

Arguments
- **program_name**
  The name of the program to be stepped.
- **task_number**
  The number of the task with the program to be stepped. Range: [1,14].

Example **>> STEPLINE "conveyor"**

Example **>> STEPLINE "maths",2**

See also **RUN**, **SELECT**, **STOP**, **TROFF**, **TRON**.

4-2-231 STOP

Type Program command

Syntax **STOP ["program_name" [, task_number]**

Description The **STOP** command will halt execution of the program specified with **program_name**. If the program name is omitted, then the currently selected program will be halted. The program name can also be specified without quotes.
In case of multiple executions of a single program on different tasks the **task_number** can be used to specify the specific task to be stopped.

Arguments
- **program_name**
  The name of the program to be stopped.
- **task_number**
  The number of the task with the program to be stopped. Range: [1,14].

Example **>> STOP progname**

Example The lines from label on will not be executed in this example.

```
STOP
label:
  PRINT var
  RETURN
```

See also **HALT**, **RUN**, **SELECT**.

4-2-232 SYSTEM_ERROR

Type System parameter (read only)

Syntax **SYSTEM_ERROR**

Description The **SYSTEM_ERROR** parameter contains system errors that occurred in the Trajexia system since the last time it was initialized. The bits in the **SYSTEM_ERROR** parameter are given in the table below.

Bit	Description
0	BASIC error
1	Battery low error
2 - 7	Reserved for future use
8	Configuration unit error (Any unit in the system)
9	Configuration device error (Any device in the system)
10 - 15	Reserved for future use
16	Unit lost error (Any unit in the system)
17	Terminator not fitted
18	Device lost error (Any device in the system)

Arguments N/A.

Example No example.

See also N/A

4-2-233 T_REF

Type Axis parameter

Syntax **T_REF**

Description The **T_REF** parameter contains the torque reference value which will be applied to the servo motor. The range of this parameter is defined by the number of available bits. For MECHATROLINK-II axes, **T_REF** takes 32 bits, so the available range is [-2147483648, 2147483648], which corresponds to a voltage range [-10V, 10V]. For Flexible axis axes, **T_REF** takes 16 bits, so available range is [-32768, 32767], which corresponds to a voltage range [-10V, 10V]. These ranges can be limited by using the **OUTLIMIT** parameter.
The actual torque reference is depending on the servo motor.

Arguments N/A

Example **T_REF AXIS(0)=1000**

See also **AXIS**, **S_REF**.

4-2-234 TABLE

Type System command

Syntax **TABLE(address, value {, value})**
TABLE(address)

Description The **TABLE** command loads data to and reads data from the TABLE array. The TABLE has a maximum length of 64000 elements. The TABLE values are floating-point numbers with fractions. The TABLE can also be used to hold information, as an alternative to variables. The **TABLE** command has two forms.

- **TABLE(address, value{, value})** writes a sequence of values to the TABLE array. The location of the first element to write is specified by address. The sequence can have a maximum length of 20 elements.
- **TABLE(address)** returns the TABLE value at the entry specified by address.

A value in the TABLE can be read-only if a value of that number or higher has been previously written to the TABLE. For example, printing **TABLE(1001)** will produce an error message if the highest TABLE location previously written to the TABLE is location 1000. The total TABLE size is indicated by the **TSIZE** parameter. Note that this value is one more than the highest defined element address.The TABLE can be deleted with by using **DEL "TABLE"** or **NEW "TABLE"** on the command line.

Notes:

- Applications like the **CAM** command, **CAMBOX** command and the **SCOPE** command in Trajexia Studio all use the same TABLE as the data area. Do not use the same data area range for different purposes.
- The TABLE and VR data can be accessed from all different running tasks. To avoid problems of two program tasks writing unexpectedly to one global variable, write the programs in such a way that only one program writes to the global variable at a time.

Arguments

- **address**
  The first location in the TABLE to read or write. Range: [0,63999]
- **value**
  The value to write at the given location and at subsequent locations.

Example **TABLE(100,0,120,250,370,470,530,550)**
The above line loads an internal table as below.

Table entry	Value
100	0
101	120
102	250
103	370
104	470
105	530
106	550

Example The following line will print the value at location 1000.
>> PRINT TABLE(1000)

See also **CAM**, **CAMBOX**, **DEL**, **NEW**, **SCOPE**, **TSIZE**, **VR**.

4-2-235 TABLEVALUES

Type	System command
Syntax	**TABLEVALUES(address, number_of_points, format)**
Description	Returns a list of TABLE points starting at the number specified. There is only one format supported at the moment, and that is comma delimited text. Note: **TABLEVALUES** is provided mainly for Trajexia Studio to allow for fast access to banks of TABLE values.
Arguments	• **address** Number of the first point to be returned • **number_of_points** Total number of points to be returned • **format** Format for the list
Example	No example.
See also	N/A

4-2-236 TAN

Type	Mathematical function
Syntax	**TAN(expression)**
Description	The **TAN** function returns the tangent of the expression. The expression is assumed to be in radians.
Arguments	• **expression** Any valid BASIC expression.
Example	**>> print TAN(PI/4)** **1.0000**
See also	N/A

4-2-237 THEN

See **IF..THEN..ELSE..ENDIF**.

4-2-238 TICKS

Type	Task parameter
Syntax	**TICKS**
Description	The **TICKS** parameter contains the current count of the task clock pulses. **TICKS** is a 32-bit counter that is decremented on each servo cycle. **TICKS** can be written and read. It can be used to measure cycles times, add time delays, etc. Each task has its own **TICKS** parameter. Use the **PROC** modifier to access the parameter for a certain task. Without **PROC** the current task will be assumed.
Arguments	N/A

Example
```
delay:
  TICKS = 3000
  OP(9,ON)
test:
  IF TICKS <= 0 THEN
    OP(9,OFF)
  ELSE
    GOTO test
  ENDIF
```

See also N/A

4-2-239 TIME

Type System parameter (read-only)

Syntax **TIME**

Description Returns the time from the real time clock. The time returned as an integer is the number of seconds since midnight 00:00:00.

Arguments N/A

Example
```
>> PRINT TIME
>> 48002.0000
```

See also N/A

4-2-240 TIME$

Type System command

Syntax **TIME$**

Description Prints the current time as defined by the real time clock as a string in 24-hour format.

Arguments N/A

Example When the time is 13:20:00
```
>> PRINT TIME$
>> 13:20:00
```

See also N/A

4-2-241 TO

See **FOR..TO..STEP..NEXT**.

4-2-242 TRANS_DPOS

Type Axis parameter (read-only)

Syntax **TRANS_DPOS**

Description Axis demand position at output of frame transformation. **TRANS_DPOS** is normally equal to **DPOS** on each axis. The frame transformation is therefore equivalent to 1:1 for each axis. For some machinery configurations it can be useful to install a frame transformation which is not 1:1, these are typically machines such as robotic arms or machines with parasitic motions on the axes. Frame transformations have to be specially written in the C language and downloaded into the controller. It is essential to contact OMRON if you want to install frame transformations.

Arguments N/A

Example No example.

See also **FRAME**.

4-2-243 TRIGGER

Type System command

Syntax **TRIGGER**

Description The **TRIGGER** command starts a previously set up **SCOPE** command. Note: Trajexia Studio uses **TRIGGER** automatically for its oscilloscope function.

Arguments N/A

Example No example.

See also **SCOPE**.

4-2-244 TROFF

Type Program command

Syntax **TROFF ["program_name"]**

Description The **TROFF** command suspends a trace at the current line and resumes normal program execution for the program specified with **program_name**. The program name can also be specified without quotes. If the program name is omitted, the selected program will be assumed.

Arguments
- **program_name**
  The name of the program for which to suspend tracing.

Example **>> TROFF "lines"**

See also **SELECT**, **TRON**.

4-2-245 TRON

Type Program command

Syntax **TRON**

Description The **TRON** command creates a breakpoint in a program that will suspend program execution at the line following the **TRON** command. The program can then for example be executed one line at a time using the **STEPLINE** command.
Notes:
- Program execution can be resumed without using the **STEPLINE** command by executing the **TROFF** command.
- The trace mode can be stopped by issuing a **STOP** or **HALT** command.

Arguments N/A

Example
```
TRON
MOVE(0,10)
MOVE(10,0)
TRON
MOVE(0,-10)
MOVE(-10,0)
```

See also **SELECT**, **TROFF**.

4-2-246 TRUE

Type Constant (read-only)

Syntax **TRUE**

Description The **TRUE** constant returns the numerical value -1.

Arguments N/A

Example
```
test:
  t = IN(0) AND IN(2)
  IF t = TRUE THEN
    PRINT "Inputs are ON"
  ENDIF
```

See also N/A

4-2-247 TSIZE

Type System parameter (read-only)

Syntax **TSIZE**

Description The **TSIZE** parameter returns the size of the TABLE array, which is one more than the currently highest defined TABLE element.
TSIZE is reset to 0 when the TABLE array is deleted using **DEL "TABLE"** or **NEW "TABLE"** on the command line.

Arguments N/A

Example The following example assumes that no location higher than 1000 has been written to the TABLE array.
```
>> TABLE(1000,3400)
>> PRINT TSIZE
1001.0000
```

See also **DEL**, **NEW**, **TABLE**.

4-2-248 UNITS

Type Axis parameter

Syntax **UNITS**

Description The **UNITS** parameter contains the unit conversion factor. The unit conversion factor enables the user to define a more convenient user unit like m, mm or motor revolutions by specifying the amount of encoder edges to include in a user unit.
Axis parameters like speed, acceleration, deceleration and the Axis commands are specified in these user units.
Note: The **UNITS** parameter can be any non-zero value, but it is recommended to design systems with an integer number of encoder pulses per user unit. Changing **UNITS** will affect all axis parameters which are dependent on **UNITS** in order to keep the same dynamics for the system.

Arguments N/A

Example A leads crew arrangement has a 5mm pitch and a 1,000-pulse/rev encoder. The units must be set to allow moves to be specified in mm.
The 1,000 pulses/rev will generate 1,000 x 4 = 4,000 edges/rev. One rev is equal to 5mm. Therefore, there are 4,000/5 = 800 edges/mm. **UNITS** is thus set as following.
>> UNITS = 1000*4/5

See also **AXIS**, **ENCODER_RATIO**.

4-2-249 UNLOCK

See **LOCK**.

4-2-250 UNTIL

See **REPEAT..UNTIL**.

4-2-251 VERIFY

Type Axis parameter

Syntax **VERIFY**

Description The verify axis parameter is used to select different modes of operation on a stepper encoder axis.

- **VERIFY=OFF**
  Encoder count circuit is connected to the **STEP** and **DIRECTION** hardware signals so that these are counted as if they were encoder signals. This is particularly useful for registration as the registration circuit can therefore function on a stepper axis.
- **VERIFY=ON**
  Encoder circuit is connected to external A,B, Z signal

Note: On the Flexible Axis, when **VERIFY=OFF**, the encoder counting circuit is configured to accept **STEP** and **DIRECTION** signals hard wired to the encoder A and B inputs. If **VERIFY=ON**, the encoder circuit is configured for the usual quadrature input.
Make sure that the encoder inputs do not exceed 5 volts.

Arguments N/A

Example **VERIFY AXIS(3)=ON**

See also N/A

4-2-252 VERSION

Type System parameter (read-only)

Syntax **VERSION**

Description The **VERSION** parameter returns the current firmware version number of the current system installed in the CJ1W-MCH72.

Arguments N/A

Example
```
>> PRINT VERSION
1.6100
```

See also N/A

4-2-253 VFF_GAIN

Type Axis parameter

Syntax **VFF_GAIN**

Description The **VFF_GAIN** parameter contains the speed feed forward gain. The speed feed forward output contribution is calculated by multiplying the change in demand position with the **VFF_GAIN** parameter value. The default value is 0.
Adding speed feed forward gain to a system decreases the Following Error during a move by increasing the output proportionally with the speed.
Note: In order to avoid any instability the servo gains should be changed only when the **SERVO** is off.

Arguments N/A

Example No example.

See also **D_GAIN**, **I_GAIN**, **OV_GAIN**, **P_GAIN**.

4-2-254 VP_SPEED

Type Axis parameter (read-only)

Syntax **VP_SPEED**

Description The **VP_SPEED** parameter contains the speed profile speed in user units/s. The speed profile speed is an internal speed which is accelerated and decelerated as the movement is profiled.

Arguments N/A

Example
```
' Wait until at command speed
MOVE(100)
WAIT UNTIL SPEED = VP_SPEED
```

See also **AXIS**, **MSPEED**, **UNITS**.

4-2-255 VR

Type System command

Syntax **VR(address)**

Description The VR command reads or writes the value of a global (VR) variable. These VR variables hold real numbers and can be easily used as an element or as an array of elements. The CJ1W-MCH72 has in total 1024 VR variables.
The VR variables can be used for several purposes in BASIC programming. The VR variables are globally shared between tasks and can be used for communications between tasks. VR variable memory area is battery backed, so all VR variables retain their values between power ups
Notes:

- The TABLE and VR data can be accessed from all different running tasks. To avoid problems of two program tasks writing unexpectedly to one global variable, write the programs in such a way that only one program writes to the global variable at a time.

Arguments

- **address**
  The address of the VR variable. Range: [0,1023].

Example In the following example, the value 1.2555 is placed into VR variable 15. The local variable **val** is used to name the global variable locally:

```
val = 15
VR(val) = 1.2555
```

Example A transfer gantry has 10 put down positions in a row. Each position may at any time be full or empty. VR(101) to VR(110) are used to hold an array of ten 1 ' s and 0 ' s to signal that the positions are full (1) or empty (0). The gantry puts the load down in the first free position. Part of the program to achieve this would be as follows:

```
movep:
  MOVEABS(115) ' Move to first put down position
  FOR VR(0) = 101 TO 110
    IF (VR(VR(0)) = 0) THEN GOSUB load
    MOVE(200) ' 200 is spacing between positions
  NEXT VR(0)
  PRINT "All positions are full"
  WAIT UNTIL IN(3) = ON
  GOTO movep

load: ' Put load in position and mark array
  OP(15,OFF)
  VR(VR(0)) = 1
  RETURN
```

The variables are backed up by a battery so the program here could be designed to store the state of the machine when the power is off. It would of course be necessary to provide a means of resetting completely following manual intervention.

Example

```
loop: ' Assign VR(65) to VR(0) multiplied by axis 1 measured position
  VR(65) = VR(0)*MPOS AXIS(1)
  PRINT VR(65)
  GOTO loop
```

See also **CLEAR_BIT**, **READ_BIT**, **SET_BIT**, **TABLE**.

4-2-256 VRSTRING

Type System command

Syntax **VRSTRING(vr_start)**

Description Combines the contents of an array of VR() variables so that they can be printed as a text string. All printable characters will be output and the string will terminate at the first null character found. (i.e. VR(n) contains 0)

Arguments
- **vr_start**
  number of first VR() in the character array.

Example **PRINT #5,VRSTRING(100)**

See also N/A

4-2-257 WA

Type System command

Syntax **WA(time)**

Description The **WA** command pauses program execution for the number of milliseconds specified for time. The command can only be used in a program.

Arguments
- **time**
  The number of milliseconds to hold program execution.

Example The following lines would turn ON output 7 two seconds after turning off output 1.
OP(1,OFF)
WA(2000)
OP(7,ON)

See also N/A

4-2-258 WAIT IDLE

Type System command

Syntax **WAIT IDLE**

Description The **WAIT IDLE** command suspends program execution until the base axis has finished executing its current move and any buffered move. The command can only be used in a program. **WAIT IDLE** works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.
Note: The execution of **WAIT IDLE** does not necessarily mean that the axis will be stationary in a servo motor system.

Arguments N/A

Example **MOVE(1000)**
WAIT IDLE
PRINT "Move Done"
The print statement is printed at the end of the movement.

Example **MOVE(1000)**
WAIT UNTIL MTYPE=0
PRINT "Movement finished"
The print statement is printed, most of the times BEFORE the movement starts, and sometimes, when the movement is finished.

Explanation Motion programs and motion sequence work in parallel and unsynchronized. One complete cycle can occur before the movement is loaded into the buffer. The program executes **MOVE(1000)** but the movement is not loaded to the buffer until the start of the next "motion sequence" so when you check **MTYPE=0**, it is **0** because the movement HAS NOT STARTED YET, not because it has finished.

See also **AXIS**, **WAIT LOADED**.

Note **WAIT IDLE** is a command specifically designed to wait until the previous movement has been finished so, it handles the delay from when the previous command is executed in the program until the command is correctly loaded in the motion buffer.

4-2-259 WAIT LOADED

Type System command

Syntax **WAIT LOADED**

Description The **WAIT LOADED** command suspends program execution until the base axis has no moves buffered ahead other than the currently executing move. The command can only be used in a program.
This is useful for activating events at the beginning of a move, or at the end when multiple moves are buffered together.
WAIT LOADED works on the default basis axis (set with **BASE**) unless **AXIS** is used to specify a temporary base axis.

Arguments N/A

Example
```
' Switch output 8 ON at start of MOVE(500) and OFF at end
MOVE(800)
MOVE(500)
WAIT LOADED
OP(8,ON)
MOVE(400)
WAIT LOADED
OP(8,OFF)
```

See also **AXIS**, **WAIT IDLE**

4-2-260 WAIT UNTIL

Type System command

Syntax **WAIT UNTIL condition**

Description The **WAIT UNTIL** command repeatedly evaluates the condition until it is **TRUE**. After this program execution will continue. The command can only be used in a program.

Arguments
- **condition**
  Any valid BASIC logical expression.

Example In this example, the program waits until the measured position on axis 0 exceeds 150, and then starts a movement on axis 1.
WAIT UNTIL MPOS AXIS(0)>150
MOVE(100) AXIS(1)

Example The expressions evaluated can be as complex as you like provided they follow BASIC syntax, for example:
WAIT UNTIL DPOS AXIS(2) <= 0 OR IN(1) = ON
The above line would wait until the demand position of axis 2 is less than or equal to 0 or input 1 is on.

See also N/A

4-2-261 WDOG

Type System parameter

Syntax **WDOG**

Description The **WDOG** parameter contains the software switch which enables the Servo Driver using the **RUN** (Servo on) input signal. The enabled Servo Driver will control the servo motor depending on the speed and torque reference values. **WDOG** can be turned on and off under program control, on Command Line Terminal.
The Servo Driver will automatically be disabled when a **MOTION_ERROR** occurs. A motion error occurs when the **AXISSTATUS** state for one of the axes matches the **ERRORMASK** setting. In this case the software switch (**WDOG**) will be turned off, the **MOTION_ERROR** parameter will have value different than 0 and the **ERROR_AXIS** parameter will contain the number of the first axis to have the error.

Arguments N/A

Example No example.

See also **AXISSTATUS**, **ERROR_AXIS**, **ERRORMASK**, **MOTION_ERROR**, **SERVO**.

4-2-262 WHILE..WEND

Type Program control command

Syntax **WHILE condition**
commands
WEND

Description The **WHILE ... WEND** structure allows the program segment between the **WHILE** and the **WEND** statement to be repeated a number of times until the condition becomes **FALSE**. In that case program execution will continue after **WEND**.
Note: **WHILE ... WEND** loops can be nested without limit.

Arguments
- **condition**
  Any valid logical BASIC expression.

Example
```
WHILE IN(12) = OFF
  MOVE(200)
  WAIT IDLE
  OP(10,OFF)
  MOVE(-200)
  WAIT IDLE
  OP(10,ON)
WEND
```

See also **FOR..TO..STEP..NEXT**, **REPEAT..UNTIL**

4-2-263 XOR

Type Mathematical operation

Syntax **expression1 XOR expression2**

Description The **XOR** (eXclusive OR) operator performs the logical **XOR** function between corresponding bits of the integer parts of two valid BASIC expressions.
The logical **XOR** function between two bits is defined as in the table below.

Bit 1	Bit 2	Result
0	0	0
0	1	1
1	0	1
1	1	0

Arguments
- **expression1**
  Any valid BASIC expression.
- **expression2**
  Any valid BASIC expression.

Example **VR(0)=10 XOR 18**
The **XOR** is a bit operator and so the binary action taking place is as follows: **01010 XOR 10010 = 11000**. The result is therefore 24.

See also N/A

SECTION 5
Examples

This chapter gives 2 categories of examples and tips:
- How-to's.
- Practical examples.

5-1 How-to's

5-1-1 Startup program

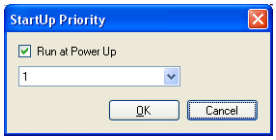
The purpose of this program is to compare the detected MECHATROLINK-II configuration with the expected one (the expected configuration is the configuration existing in the moment you create the program).
The STARTUP program does these actions:
- Checks the number of nodes in the system.
- Checks that the node numbers agrees.
- Checks if all slaves are connected and have power.
- Any non agreement, the program stops.
- Sets the correct **ATYPE** as selected in the intelligent axis window.
- Sets the mode, **Run** or **Commisioning**.

5-1-1-1 How to set a startup program

When you add a new CJ1W-MCH72 device to the solution in Trajexia Studio, 2 programs are created by default: the SHELL program, and an application program called APPLICATION.

Default programs SHELL and APPLICATION

A CJ1W-MCH72 device can execute a program at startup: when the device is switched on, it executes the program.
You can set the startup priority for a program in Trajexia Studio with the **Priority** property in the **Properties** window. If you click the ellipsis button in the edit field of this property, the **StartUp Priority** window shows.



StartUp Priority window

To set the program to run at power up, select the **Run at Power Up** check box and select a priority in the list. Possible priority values are **Default** or 1 (lowest priority) to 14 (highest priority).
To set the program not to run at startup, clear the **Run at Power Up** check box.

Note The SHELL program by default runs at startup at priority 1.

Note OMRON recommends that the statement **RUN "your_program"** is used at the end of the Startup program to start your application program. The application program starts when the startup program is executed successfully and without errors.
If you set an application program to "Run at startup" there is a risk that the machine starts if there is an error on the MECHATROLINK-II bus.

5-1-1-2 Example

```
'=============================================
'THE FIRST PART OF THE PROGRAM
'CONSISTS OF A CHECK SEQUENCE TO
'VERIFY THAT THE DETECTED AXIS CONFIGURATION IS THE
'EXPECTED ONE.
'IF YES, THE PROGRAM FINISHES AND STARTS "SHELL".
'IF NOT, THE PROGRAM STOPS AND NO OTHER PROGRAM STARTS.
'THIS PROGRAM MUST BE SET TO RUN AT POWER UP IN 'A LOW
'PRIORITY TASK (1 IN THIS EXAMPLE)
'=============================================
'Start MECHATROLINK Section
' Check detected slaves
' Unit 0
IF NOT MECHATROLINK(0,3,0) THEN
    PRINT "Error getting slave count for unit 0"
    STOP
ELSE
    IF VR(0) <> 3 THEN
        PRINT "Incorrect slave count for unit 0"
        STOP
    ENDIF
ENDIF
IF NOT MECHATROLINK(0,4,0,0) THEN
    PRINT "Error getting address for unit 0, station 0"
    STOP
ELSE
    IF VR(0) <> 65 THEN
        PRINT "Incorrect address for unit 0, station 0"
        STOP
    ENDIF
ENDIF
IF NOT MECHATROLINK(0,4,1,0) THEN
    PRINT "Error getting address for unit 0, station 1"
    STOP
ELSE
    IF VR(0) <> 66 THEN
        PRINT "Incorrect address for unit 0, station 1"
        STOP
    ENDIF
ENDIF
IF NOT MECHATROLINK(0,4,2,0) THEN
    PRINT "Error getting address for unit 0, station 2"
    STOP
ELSE
    IF VR(0) <> 67 THEN
        PRINT "Incorrect address for unit 0, station 2"
        STOP
    ENDIF
ENDIF
' Set axis types
' Unit 0
ATYPE AXIS(0)=40
ATYPE AXIS(1)=40
ATYPE AXIS(2)=40
' Set drives into run mode
```

```
' Unit 0
MECHATROLINK(0,20,65)
MECHATROLINK(0,20,66)
MECHATROLINK(0,20,67)
'Stop MECHATROLINK Section
'================================================
'THIS SECTION MUST BE MANUALLY SET BY THE USER
'ACCORDING TO THE APPLICATION. TYPICAL ACTIONS ARE
'VARIABLE INITIALIZATION, SERVO/AXIS SETTING, NAMING
'GLOBAL VARIABLES AND START THE "SHELL" PROGRAM.
'================================================
'Define Names for global variables
GLOBAL "project_status",100
GLOBAL "alarm_status",101
GLOBAL "action",102
'Initialize variables
VR(0)=0
project_status=0
alarm_status=0
action=0
'Start SHELL program
RUN "SHELL",2
STOP
```

5-1-2 Gain settings

The gain setting is related to the mechanical system to which the motor is attached. There are three main concepts:

- Inertia ratio
- Rigidity
- Resonant frequency.

These concepts are described in the Hardware Reference Manual in the chapter System Philosophy.

This section shows example parameter values for:

- Speed Loop Gain
- Proportional position gain
- Velocity Feed Forward gain.

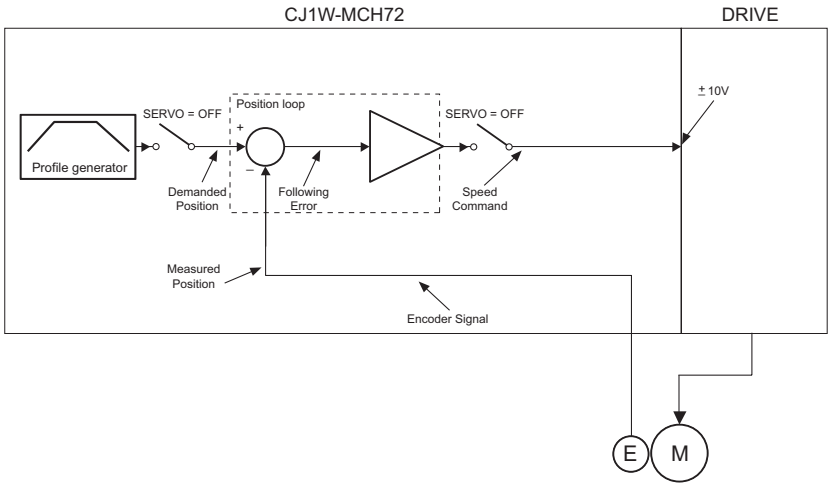
The example values for the program and motion parameters in the Trajexia system are given below. Note that they are appropriate for 13-bit encoders.

Drive Parameter value	Description
Pn103 = 716	Inertia ratio
Pn110 = 0012	No autotuning
Pn202=1	Gear ratio numerator
Pn203=1	Gear ratio denominator

Motion Parameter values	Description
UNITS =1	Working in encoder counts
SPEED=200000	Speed setting
ACCEL=1000000	Acceleration setting

Motion Parameter values	Description
DECEL=1000000	Deceleration setting
MOVEMENT=81920	10 Turns

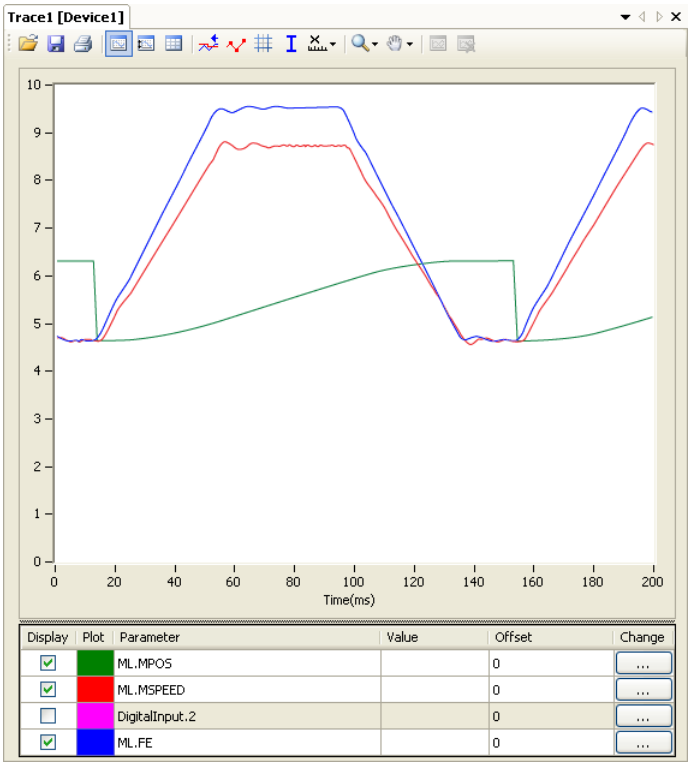
5-1-2-1 Speed mode examples



In this mode the position loop is closed in Trajexia and the Speed loop is closed in the Servo Driver. The **Speed** axis parameter is sent to the Servo Driver, and reads the position feedback.

```
BASE(0)
ATYPE=44 'Servo axis encoder mode
SERVO=1
WDOG=1
DEFPOS(0)
loop:
    MOVE(81920)
    WAIT IDLE
    WA(100)
    DEFPOS(0)
GOTO loop
```

Example 1

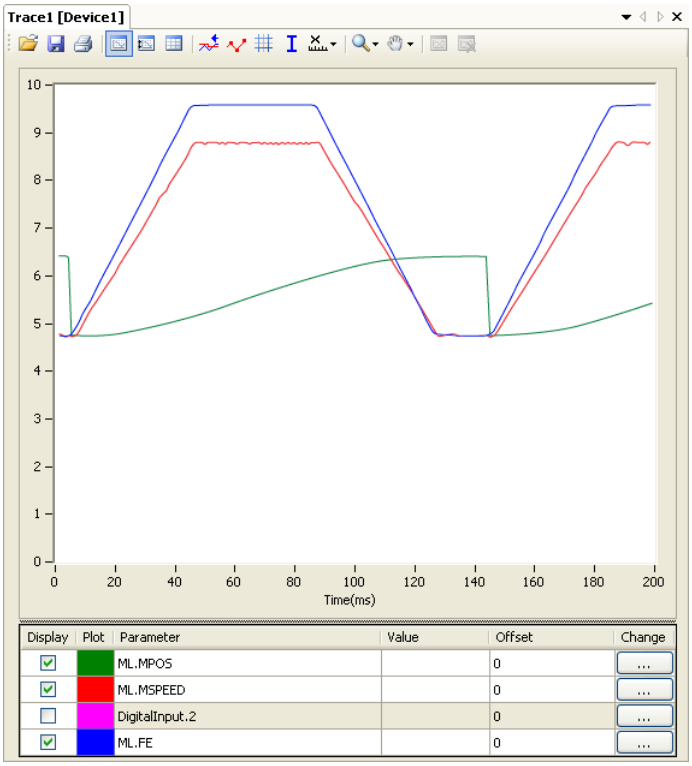


Only proportional gain has a set value, the Following Error is proportional to the speed.
The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=0
Fn001=4

Note The colours and scale of the oscilloscope for speed mode are as follows:
Red: MSPEED (Measured Axis speed). Units is 50 units/ms/division
Blue: FE (Following Error). Units is depending on the graph
Green: MPOS (Measured Axis position). 50000 units/division

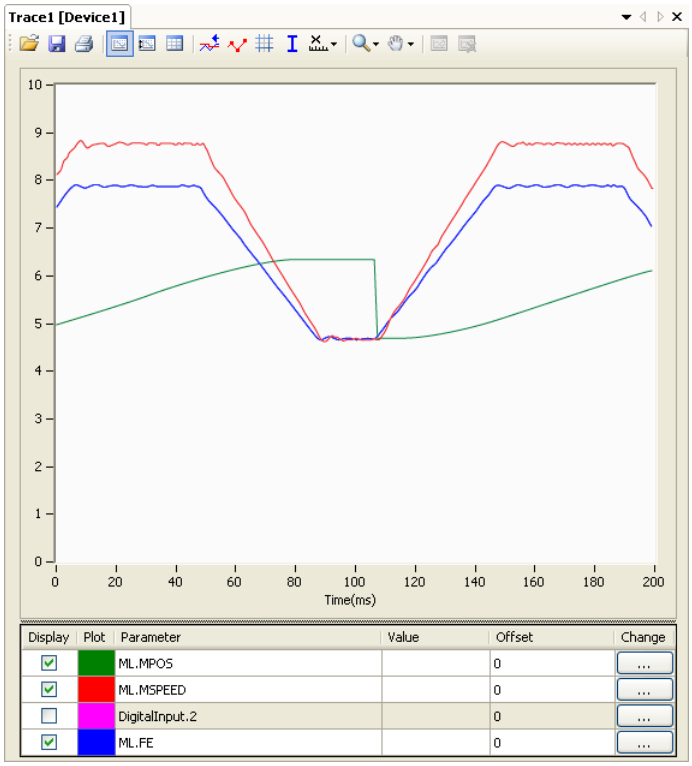
Example 2



The value for rigidity is increased. The error magnitude remains the same but the ripple, the speed stability and overshoot are better.
The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=0
Fn001=6

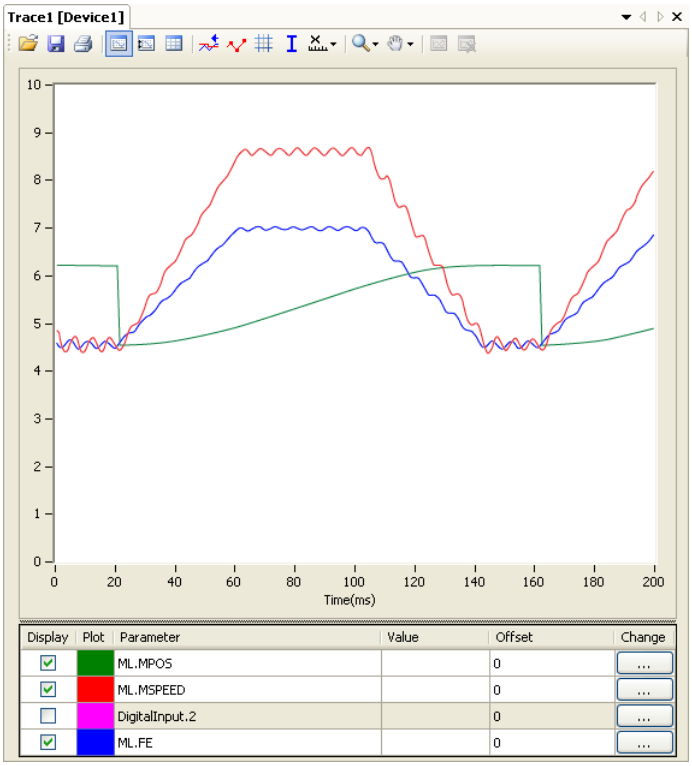
Example 3



The parameter **P_GAIN** is increased further. The Following Error decreases proportionally.
The parameter values for the example are:

Motion Parameter values
P_Gain=200000
VFF_GAIN=0
Fn001=6

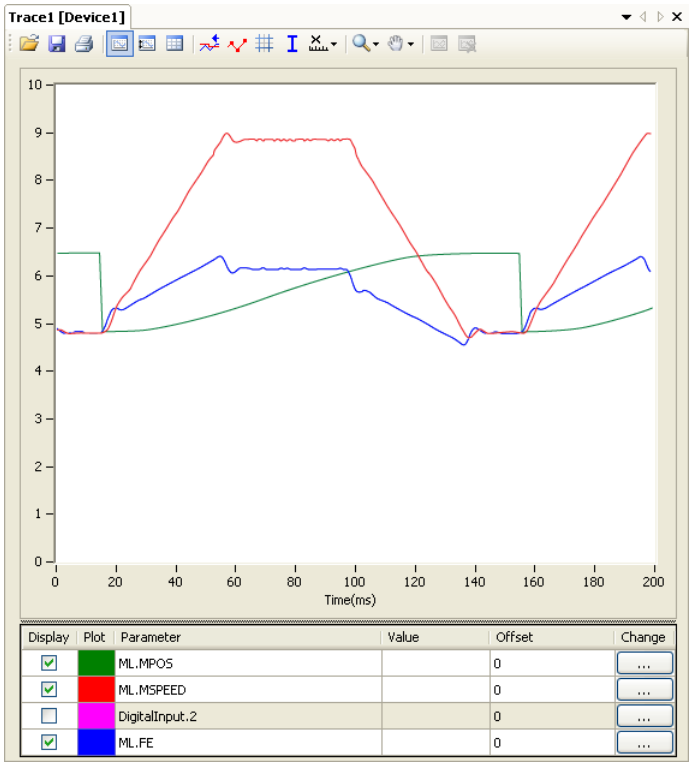
Example 4



The value of the parameter **P_GAIN** two times the value in example 1. The Following Error is half, but there is vibration due to the excessive gains.
The parameter values for the example are:

Motion Parameter values
P_Gain=262144
VFF_GAIN=0
Fn001=6

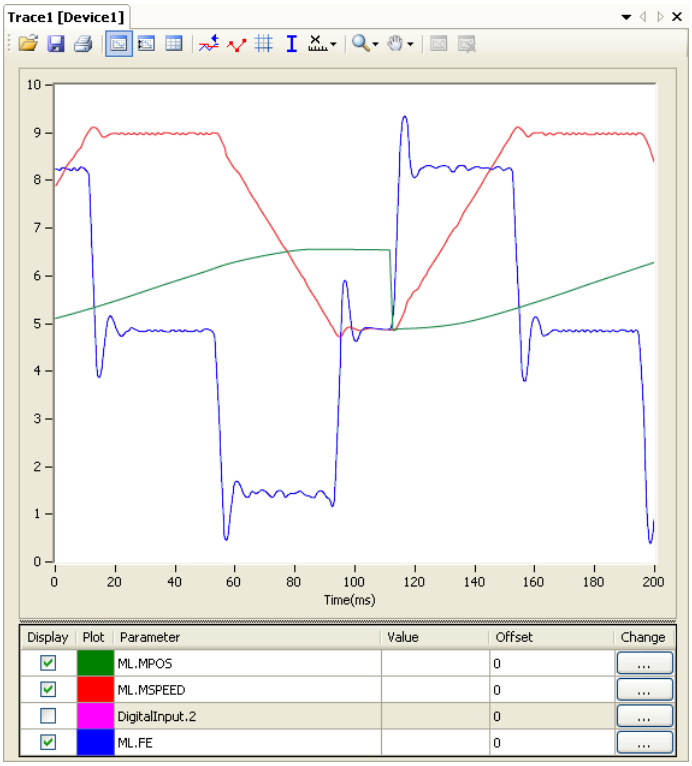
Example 5



The value of the parameter **P_GAIN** is set to the value in example 1. The value of **VFF_GAIN** is increased. The Following Error is reduced without a reduction to the stability. The Following Error is not proportional to the speed. The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=1400000
Fn001=6

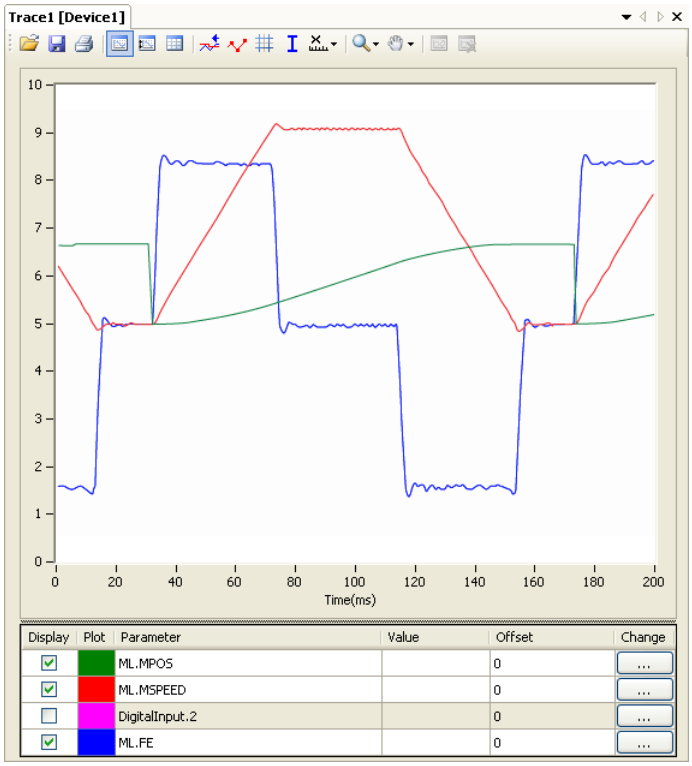
Example 6



With this value of VFF_GAIN the Following Error is proportional to the acceleration, and smaller than with just proportional gain (the scaling is 20 units/division). The Following Error approaches zero during constant speed.
The negative effect of this set of values is the overshoot and undershoot when the acceleration changes; this can be reduced but not eliminated by increasing the speed loop gain, if the mechanical system can cope with a high gain.
The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=1573500
Fn001=6

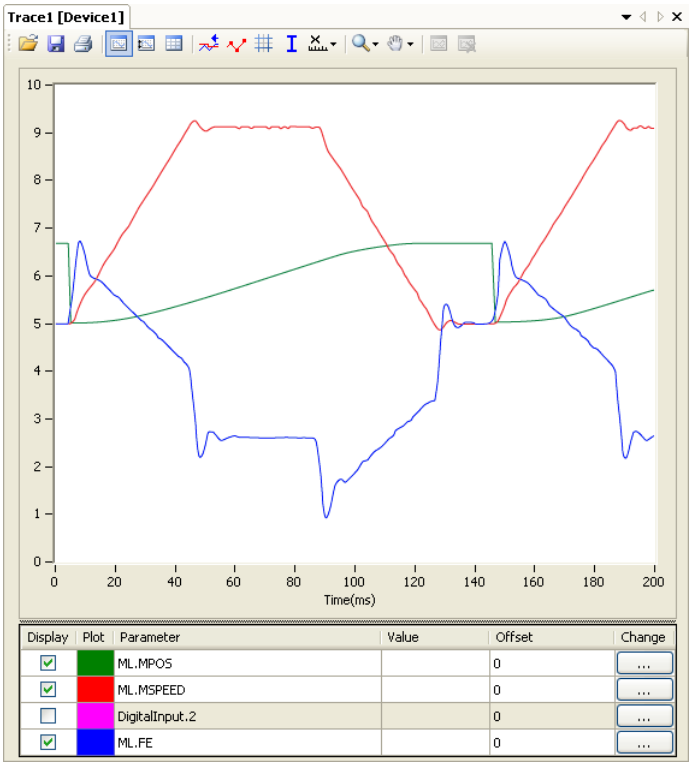
Example 7



The value of the rigidity is increased from 6 to 8. The overshoot/undershoot is smaller but the motor has more vibration.
The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=1573500
Fn001=8

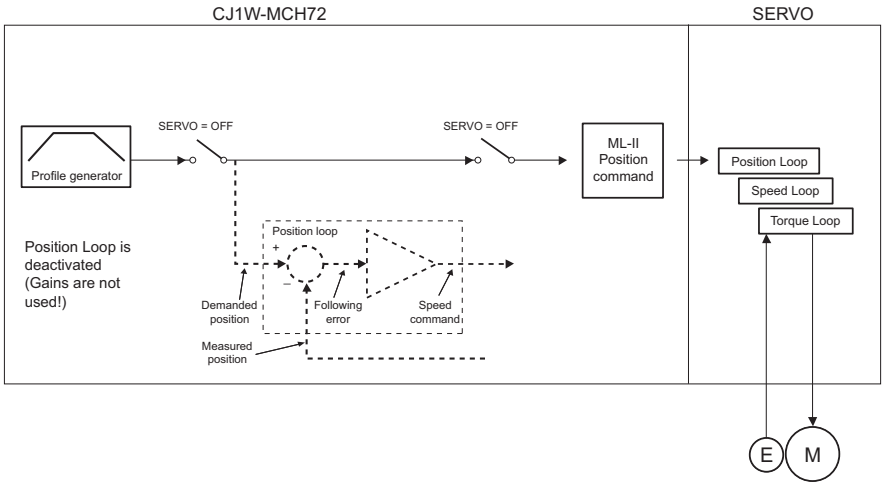
Example 8



Opposite to the P_GAIN, where the higher, the better (the limit is when the mechanical system starts vibrating), for the VFF_GAIN there is an optimum value (the one in test 6), values higher than this value has an error proportional to the speed/acceleration but with different sign. The required correction is too large.
The parameter values for the example are:

Motion Parameter values
P_Gain=131072
VFF_GAIN=1650000
Fn001=6

5-1-2-2 Position mode examples



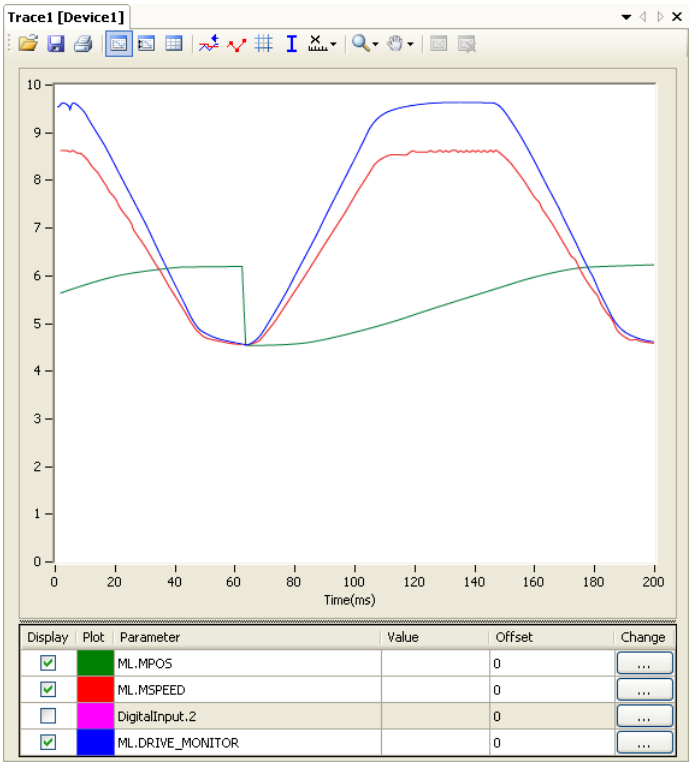
In this mode the position and speed loop are closed in the Servo Driver. The CJ1W-MCH72 sends the position command through the MECHATROLINK-II network to the Servo Driver, and reads the position feedback.
Note that this system has no sample delay as compared to the position loop in the Servo Driver, the Demanded Position in cycle "n" with the Measured Position in cycle "n".
The CJ1W-MCH72, for the internal handling, continues to use its own position loop, so the Following Error that read in the Axis parameter in the CJ1W-MCH72 is not the real one in the Servo Driver. To read the correct Following Error use DRIVE_MONITOR.
Adjust the rigidity of the servo, the speed loop gain and the position loop gain at the same time using just proportional position gain. The results are similar to the MECHATROLINK-II Speed mode with the advantages:

- The tuning is more simple, only the rigidity (Fn001) and, if necessary, the feedforward gain (Pn109) needs to be set.
- The position loop in the servo is faster (250µs) than in the CJ1W-MCH72 and it is turned together with the speed loop.
- There is no sample time delay between "Target position" and "Measured position".

To do a finetune the different gain parameters can be changed individually.

```
BASE(0)
ATYPE=41 'MECHATROLINK Position mode
SERVO=1
DRIVE_CONTROL=2 'To monitor the Following Error in
                'DRIVE_MONITOR
WDOG=1
DEFPOS(0)
loop:
    MOVE(81920)
    WAIT IDLE
    WA(100)
    DEFPOS(0)
GOTO loop
```

Example 1



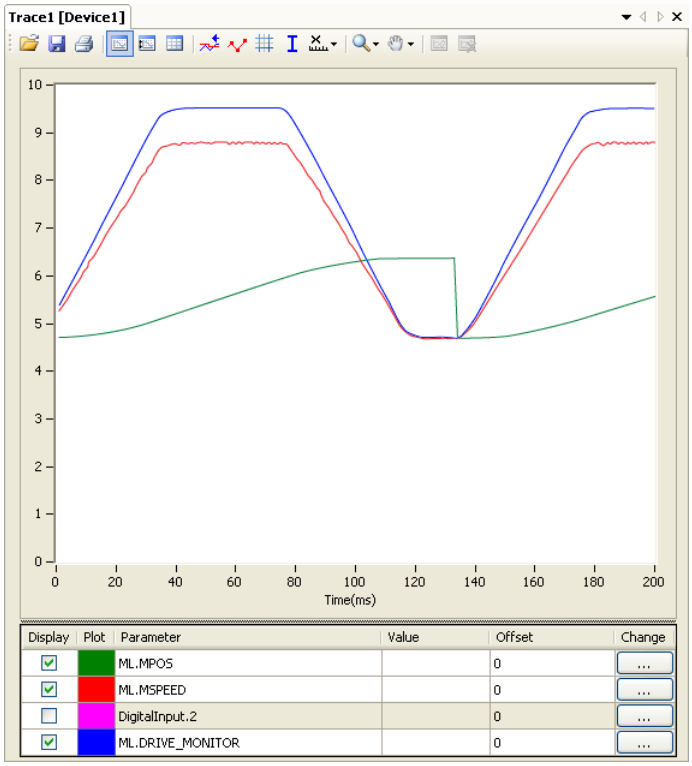
The Following Error is proportional to the speed. There is a "soft profile" due to the low rigidity setting (low gain).

Note The colours and scale of the oscilloscope for position mode are as follows:
Red: MSPEED (Measured Axis speed). Units is 50 units/ms/division
Blue: DRIVE_MONITOR (set as Following Error in the Servo Driver). Units is depending on the graph
Green: MPOS (Measured Axis position). 50000 units/division

The parameter values for the example are:

Motion Parameter values
Fn001=4
Pn109=0

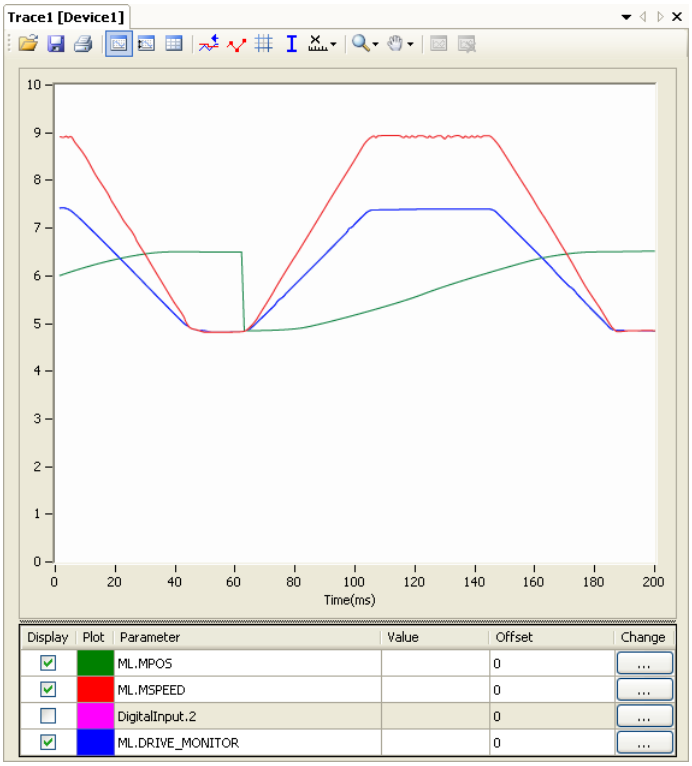
Example 2



The Following Error reduces as the rigidity increases.
The parameter values for the example are:

Motion Parameter values
Fn001=6
Pn109=0

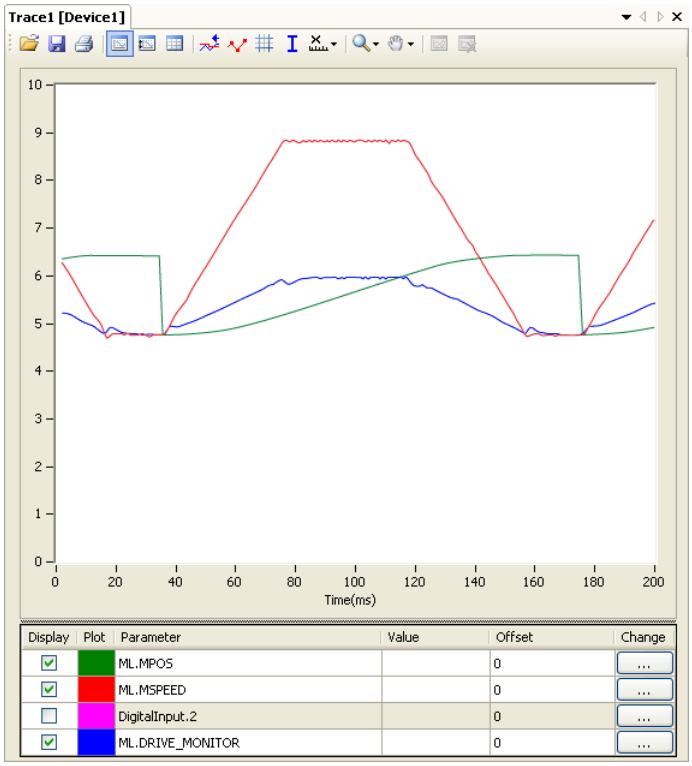
Example 3



With high gain the motor starts to vibrate but the profile is more stable that in MECHATROLINK-II Speed mode.
The parameter values for the example are:

Motion Parameter values
Fn001=8
Pn109=0

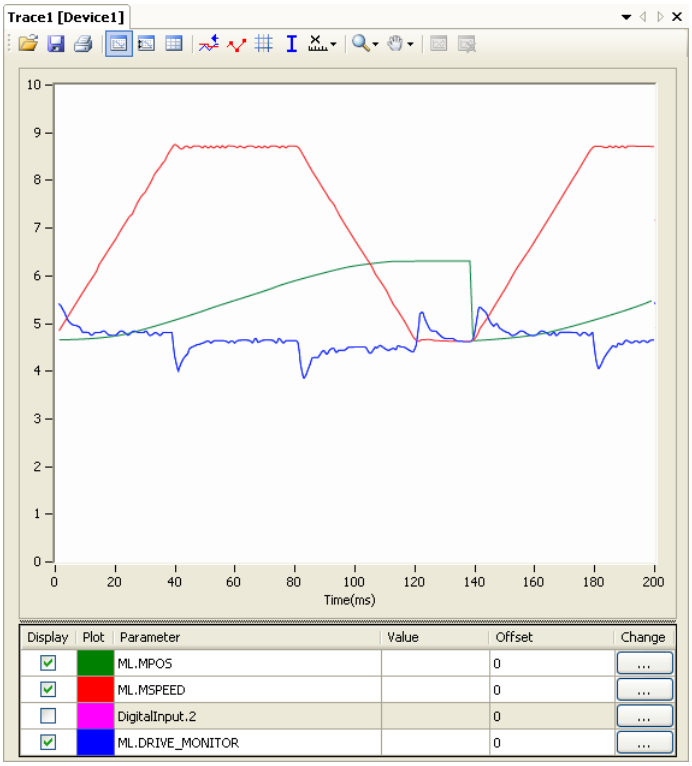
Example 4



The effect of the Feedforward gain is that the Following Error is reduced and the effect is proportional to the acceleration.
The parameter values for the example are:

Motion Parameter values
Fn001=6
Pn109=95

Example 5



With the feedforward set to 100%, the Following Error is very small and proportional to the acceleration. The optimum value of 100% correction is the maximum value that can be set. The parameter value of Pn109 is easier to set than the parameter value of VFF_GAIN.
The parameter values for the example are:

Motion Parameter values
Fn001=6
Pn109=100

5-1-3 Setting the UNITS axis parameter and gear ratio

In controlling the mechanical axis with the CJ1W-MCH72, a Servo Driver and a servo motor, the only measurement units that the hardware understands are encoder counts. All commands to the driver to move an axis are expressed in encoder counts. All feedback information about axis positions is also expressed in encoder counts. When writing programs in BASIC to achieve movements or a sequence of movements, a user can prefer to work with user defined units, such as millimeter, centimeter, meter, degree of angle, "product", "rotation", "stations". The **UNITS** axis parameter contains the conversion factor between encoder counts and user defined units. All axis parameters related to motion and arguments of axis commands that determine the amount of motion are expressed in these user units. This parameter enables the user to define the most convenient units to work with. For example, for a moving part that makes a linear motion, you can prefer mm, or fraction of mm. For a moving part that makes a rotation motion, you can prefer a degree of angle or its fraction. For more information on the **UNITS** axis parameter, see section 4-2-248.

However, the user must be aware that not only the **UNITS** axis parameter matters in the conversion between encoder counts and user defined units. Certain Servo Driver parameters and some characteristics of the mechanical system are also important. The following sections describe which Servo Driver parameters are important for this conversion. We also give examples of how to set those parameters and the **UNITS** axis parameter, taking the characteristics of the mechanical system into account.

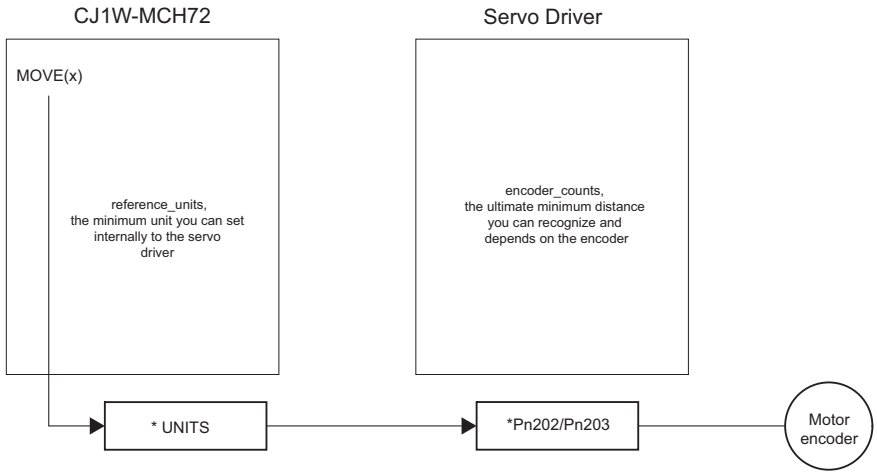
5-1-3-1 Conversion between encoder counts and user defined units

Two very important parameters of the Servo Drivers for conversion of encoder counts into user units are the electronic gear ratio numerator and the electronic gear ratio denominator. The table below gives these parameters for the Servo Drivers.

Servo Driver	Numerator	Denominator
Sigma-II	Pn202	Pn203
Sigma-V	Pn20E	Pn210
Junma	Pn20E	Pn210

Note The remainder of this section uses the parameters of the Sigma-II Servo Driver, that is, Pn202 and Pn203. If you use a Sigma-V or a Junma Servo Driver, you must use the corresponding parameters.

If a servo motor with an absolute encoder is used, setting parameter Pn205 (Multiturn limit) is also necessary.



Parameter Pn202 is the electronic gear ratio denominator (G1). Parameter Pn203 is the electronic gear ratio numerator (G2). The servo motor rotates using the value of the position command signal sent by the CJ1W-MCH72, multiplied by the electronic gear (Pn202, Pn203). On the output (servo motor) side, the signal is expressed in number of encoder pulses. For more information on Servo Driver parameters Pn202 and Pn203, see the Sigma-II Servo Driver manual.

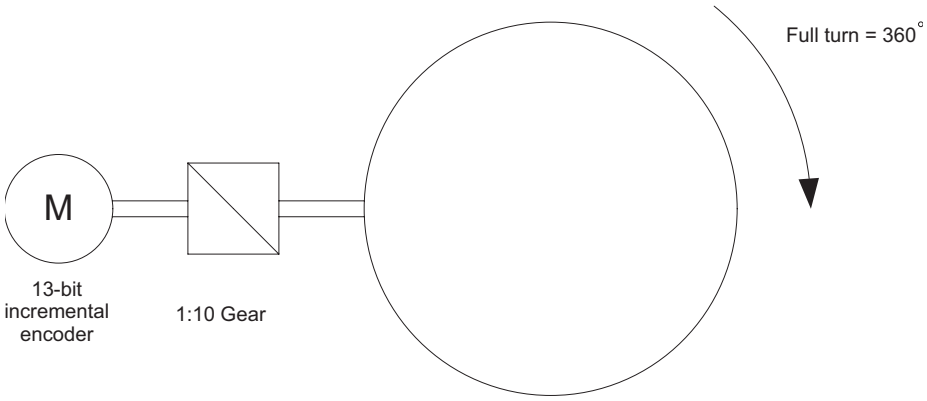
The UNITS axis parameter effectively expresses the ratio between user units that the user wants to use in the program and the position sent to the Servo Driver via the MECHATROLINK-II bus. Taking the electronic gear setting into

account, the equation expressing the relation between user units, the **UNITS** parameter, parameters Pn202 and Pn203, encoder pulses and mechanical measurement units is:

$$\frac{Pn202}{Pn203} \cdot UNITS = \frac{y \cdot encoder\_counts}{x \cdot user\_units}$$

where *y* is the number of encoder counts and *x* is the amount in user units.

5-1-3-2 Example 1



The mechanical system consists of a simple rotary table. A servo motor with 13-bit incremental encoder is used. The gear ratio of the gearbox is 1:10.
The desired user units are degree of angle. This system can be described with the following equations:

$$1 \cdot motor\_revolution = 2^{13} \cdot encoder\_counts$$

$$10 \cdot motor\_revolution = 1 \cdot machine\_cycle$$

$$1 \cdot machine\_cycle = 360^{o}$$

The combination of these equations results in:

$$\frac{Pn202}{Pn203} \cdot UNITS = \frac{2^{13} \cdot encoder\_counts}{1 \cdot motor\_revolution} \frac{10 \cdot motor\_revolution}{1 \cdot machine\_revolution} \frac{1 \cdot machine\_revolution}{360^{o}} =$$

$$\frac{2^{13} \cdot 10}{360} \frac{encoder\_counts}{degree}$$

And therefore:

$$\frac{Pn202}{Pn203} \cdot UNITS = \frac{2^{13} \cdot 10}{360}$$

From this equation, we can derive the values for Pn202, Pn203 and **UNITS**, given the following restrictions and recommendations:

1 Pn202 and Pn203 are integers.
2 UNITS must not have an infinite number of decimal digits. This can create rounding errors that result in small position errors that add up to large accumulative position errors.
3 For reasons of stability, it is necessary to avoid situations where Pn202/Pn203 is less than 0.01 or greater than 100. It is recommended that Pn202/Pn203 is approximately 1.

We can now rewrite the last equation to:

$$UNITS \cdot \frac{Pn202}{Pn203} = 2^{13} \frac{10}{360}$$

One solution to this equation is:

$$UNITS = 2^{13} = 8192$$
$$Pn202 = 10$$
$$Pn203 = 360$$

When we consider the third recommendation from the above list (avoid situations where Pn202/Pn203 is less than 0.01 or greater than 100), we can rewrite the last equation to:

$$UNITS \cdot \frac{Pn202}{Pn203} = 2^{13} \frac{10}{360} = 2^{8} \frac{2^{5}}{36} = 2^{8} \frac{32}{36}$$

This gives us the solution:

$$UNITS = 2^{8} = 256$$
$$Pn202 = 32$$
$$Pn203 = 36$$

With these values, the command **MOVE(28)** rotates the table 28 degrees in positive direction.

5-1-3-3 Absolute encoder setting

The absolute encoder keeps the current motor position, even if there is no power supplied. The absolute encoder gives the position within one turn (that is, a fraction from 0 to and excluding 1), and it has a multiturn counter. You can set the multiturn behaviour of the absolute encoder with the parameter Pn205 of the Sigma-II Servo Driver. This parameter adjusts the maximum number of turns that the counter counts before it has an overflow. For more information on Servo Driver parameter Pn205, see the Sigma-II Servo Driver manual. Taking this parameter value into account, the maximum position value the encoder can signal is:

$$max\_encoder\_count\_value = (Pn205 + 1) \cdot encoder\_counts - 1$$

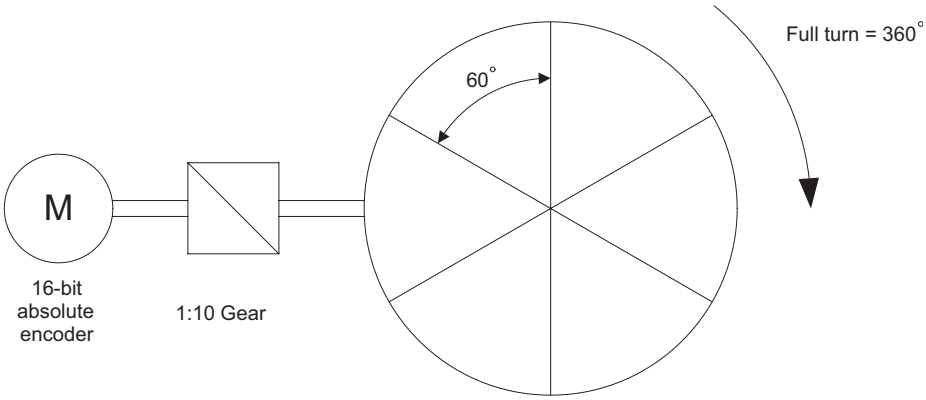
which makes it Pn205 complete turns, plus the position within one turn (the fraction from 0 to and excluding 1). When the MECHATROLINK-II connection is established with the drive, the absolute encoder position is read from the drive and the value is written in **MPOS** (after the conversion: **UNITS** × Pn202/Pn203). When the mechanical system has a limited travel distance to move, like in a ball screw, the value of the parameter Pn205 should be set large enough to have an overflow of the counter out of the effective position. This is

called limited axis or finite axis. A typical example of a limited axis is a ball screw, as shown in . When the mechanical system always moves in the same direction, it reaches the overflow of the multiturn counter. In this case, the value of Pn205 must guarantee that the overflow always occurs in the same position with respect to the machine. This is called unlimited axis and a typical example of it is a turntable shown in . It can be achieved with the following equation: the smallest value of *m* such that:

$$n \cdot machine\_cycles = m \cdot motor\_revolution$$

Because *n* and *m* are integers: Pn205 = *m* – 1. This setting is explained in the following example.

5-1-3-4 Example 2



The mechanical system consists of simple rotary table shown in the figure. A servo motor with 16-bit absolute encoder is used. The gear ratio of the gearbox is 1:10. The desired user units are degree of angle. The rotary table is divided in six sections of 60 degrees each. Therefore the machine_cycle is 60 degrees.
When we apply the last equation to the above, we get:

$$10 \cdot motor\_revolution = 1 \cdot machine\_revolution = 6 \cdot machine\_cycle$$

Simplification of this equation gives:

$$5 \cdot motor\_revolution = 3 \cdot machine\_cycle$$

This results in:

$$Pn205 = 5\text{-}1 = 4$$

We calculate the parameters as we did in example 1. This gives:

$$UNITS = 2^{11} = 2048$$
$$Pn202 = 32$$
$$Pn203 = 36$$

To guarantee the correct overflow both in the CJ1W-MCH72 and in the Servo Driver, we must set two additional axis parameters: **REP_DIST** = 60, and **REP_OPTION** = 1. With these settings, the command **MOVE(35)** rotates the table 35 degrees in positive direction. The range of possible **MPOS** and **DPOS** values is from 0 degrees to 60 degrees.

⚠ **Caution** You must initialize the absolute encoder before you use it for the first time, when the battery is lost during power off and when the multiturn limit setting in the parameter Pn205 is changed. The initialization can be done on the display of the Servo Driver or with the software tool. For more detail on initialising absolute encoder, please see the Sigma-II Servo Driver manual.

⚠ **Caution** It is possible to reset the multiturn counter, but it is not possible to reset the position within one turn (the fraction from 0 to and excluding 1). To adjust zero offset, use the parameter Pn808. For more details see the NS115 MECHATROLINK-II Interface Unit manual.

⚠ **Caution** At power up, the absolute encoder position is read from the motor and written to **MPOS** using the following conversion:

- For **MPOS**:

$$\text{Absolute\_MPOS} = \text{abs\_position\_encoder} \cdot \frac{1}{\text{UNITS}} \cdot \frac{\text{Pn203}}{\text{Pn202}}$$

- This is correct if

$$(\text{Pn205} + 1) \cdot \frac{\text{Pn203}}{\text{Pn202}} \cdot \text{encoder\_counts} < 2^{24}$$

- If this value is greater than 2^{24}, **MPOS** can have incorrect values at start-up. To avoid this problem, add the program code **DEFPOS = ENCODER/UNITS** after all **UNITS** initializations.

⚠ **Caution** To make sure that the absolute position is always correct, you must make sure that

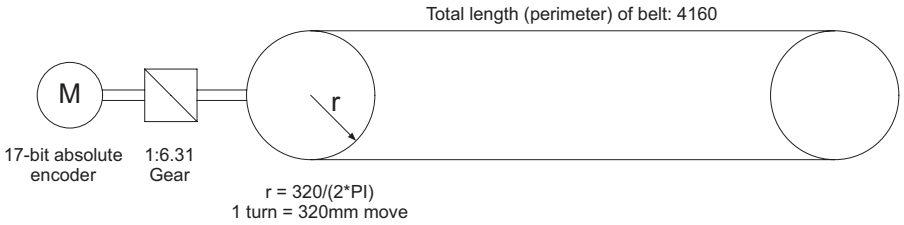
$$(\text{Pn205} + 1) \cdot \text{encoder\_resolution} < 2^{32}$$

and that

$$(\text{Pn205} + 1) \cdot \text{encoder\_resolution} \cdot \frac{\text{Pn203}}{\text{Pn202}} < 2^{32}$$

Note that this is not obvious for the high-resolution encoders of the Sigma-V motors.

5-1-3-5 Example 3



The mechanical system uses a servo motor with an 17-bit absolute encoder. The mechanical gear ratio of the gearbox is 1:6.31. One rotation of the pulley moves the moving part on the belt 320 mm. The total length of the belt, and therefore the total moving range of the motion part, is 4160 mm.
The mechanical measurement units must be mm. This means that all axis parameters and commands given to the CJ1W-MCH72 are expressed in mm. Using the same procedure as in example 1, the equation expressing the relationship between user units and encoder counts is:

$$\frac{Pn202}{Pn203}\,UNITS = \frac{2^{17}\cdot encoder\_counts}{1\cdot motor\_revolution}\,\frac{6.31\cdot motor\_revolution}{1\cdot pulley\_revolution}\,\frac{1\cdot pulley\_revolution}{320mm} =$$

$$\frac{2^{17}\cdot 6.31}{320}\,\frac{encoder\_counts}{mm}$$

Therefore:

$$\frac{Pn202}{Pn203}\,UNITS = \frac{2^{17}\cdot 6.31}{320} = \frac{2^{17}}{2^{5}}\,\frac{631}{1000} = 2^{12}\,\frac{631}{8.125} = 2^{12}\,\frac{631}{2^{3}\cdot 125} = 2^{9}\,\frac{631}{125}$$

One solution is:

$$UNITS = 2^{9} = 512$$
$$Pn202 = 631$$
$$Pn203 = 125$$

Note that we have not used the pulley radius in the calculation. This is to avoid the use of π, which cannot be expressed as a fractional number). In toothed pulleys, the number of teeth and mm per tooth is commonly used.
The calculation of the multiturn limit setting is:

$$m\cdot motor\_revolution = n\cdot machine\_cycle$$

$$m\cdot motor\_revolution = n\cdot machine\_cycle\,\frac{4160\cdot pulley\_revolution}{320\cdot machine\_cycle} = n\cdot 13\cdot pulley\_revolution$$

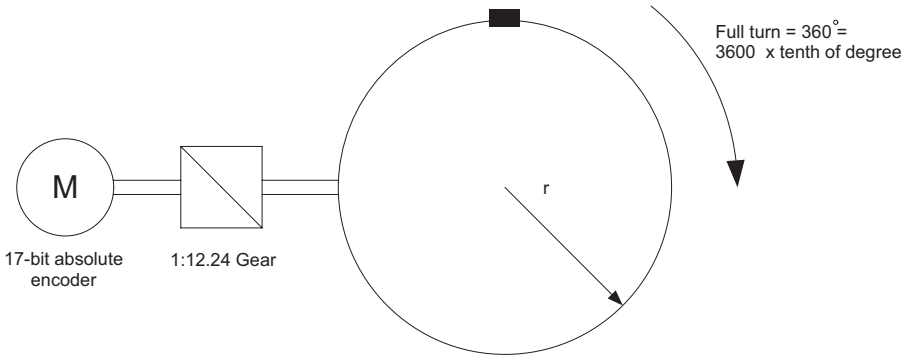
$$= n\cdot 13\,\frac{6\cdot 31\ motor\_revolution}{1\ pulley\_revolution} = n\cdot 82.03\cdot pulley\_revolution$$

$$m = n\cdot 82.03$$

The smallest integer *m* for which this equation is valid is 8203. This results in Pn205 = 8202.

In addition, to limit the motion units range to the moving range of the motion part, the following axis parameters must be set: **REP_DIST = 4260**, and **REP_OPTION = 1**. With these settings, executing **MOVE(38)** moves the moving part 38 mm in forward direction. The range of possible **MPOS** and **DPOS** values is 0 mm to 4160 mm.

5-1-3-6 Example 4



The mechanical system uses a servo motor with a 17-bit absolute encoder. The mechanical gear ratio of the gearbox is 1:12.24. The mechanical measurement units must be tenths of an angle degree. Therefore the total repeat distance for the full turn of the moving part is 3600 tenths of an angle degree.
With the same procedure as in example 1, we have:

$$\frac{Pn202}{Pn203}\ UNITS = \frac{2^{17} \cdot encoder\_counts}{1 \cdot motor\_revolution}\ \frac{12.24 \cdot motor\_revolution}{1 \cdot machine\_revolution}\ \frac{1 \cdot pulley\_revolution}{3600\ tenth\ of\ degree} =$$

$$= \frac{2^{17} \cdot 12.24}{3600}\ \frac{encoder\_counts}{tenth\ of\ degree}$$

Therefore:

$$UNITS = \frac{Pn202}{Pn203} = 2^{17}\ \frac{1224}{360000}$$

One solution is:

$$UNITS = 2^{17} = 131072$$
$$Pn202 = 1224$$
$$Pn203 = 360000$$

Because the greatest common divisor of Pn202 and Pn203 must be 1, we get: Pn202 = 17 and Pn203 = 500. Therefore, the parameters are:

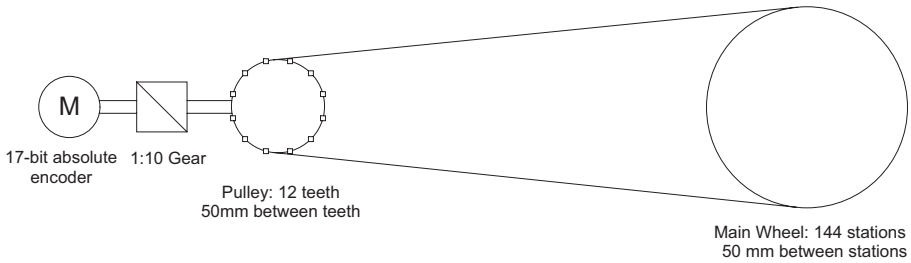
UNITS = 131072
Pn202 = 17
Pn203 = 500
Pn205 = 16
REP_DIST = 3600
REP_OPTION = 1

To calculate the multiturn limit setting Pn205, we have:

$$m \cdot motor\_revolution = n \cdot machine\_cycle = n \cdot 12.24 \cdot motor\_revolution$$

The evident solution is: *n* = 100 and *m* = 1224. Or, when we simplify the factors: *n* = 25 and *m* = 306. Therefore: Pn205 = m – 1 = 305. With these settings, executing **MOVE(180)** moves the moving part 180 tenths of an angle degree or 18 angle degrees in forward direction.

5-1-3-7 Example 5



The mechanical system uses a servo motor with a 17-bit absolute encoder. The mechanical gear ratio of the gearbox is 1:10. The pulley has got 12 teeth, and each two are 50 mm apart. One complete turn of the pulley equals 144 stations on the main wheel. The distance between two stations is 50 mm. The mechanical measurement units must mm. Total repeat distance must be the distance between two stations, 50mm.
With the same procedure as in example 1, we have:

$$\frac{Pn202}{Pn203}\ UNITS =$$

$$\frac{2^{17} \cdot encoder\_counts}{1 \cdot motor\_revolution} \cdot \frac{10 \cdot motor\_revolution}{1 \cdot pulley\_revolution} \cdot \frac{1 \cdot pulley\_revolution}{12 \cdot station} \frac{1 \cdot station}{50mm} =$$

$$= \frac{2^{17} \cdot 10}{12 \cdot 50} \frac{encoder\_counts}{mm}$$

Therefore, if we use the mechanical system to set the electronic gear ratio, we have:

$$UNITS \frac{Pn202}{Pn203} = \frac{2^{17}}{50} \frac{10}{12}$$

One possible solution is:

$$UNITS = \frac{2^{17}}{50}$$

Pn202 = 5
Pn203 = 6
Pn205 = 4

Because $2^{17}/50$ is a number with an infinite number of decimal digits, we can choose the following:

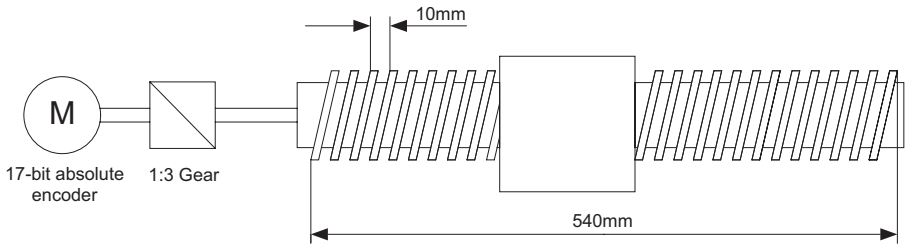
$$UNITS \frac{Pn202}{Pn203} = 2^{17} \frac{10}{50 \cdot 12} = 2^{17} \frac{10}{600} = 2^{17} \frac{1}{60} = 2^{17} \frac{1}{2^2 \cdot 15} = 2^{15} \frac{1}{15}$$

Therefore, the parameters are:

$UNITS = 2^{15} = 32768$
Pn202 = 1
Pn203 = 15
Pn205 = 4
REP_DIST = 50
REP_OPTION = 1

With these settings, executing **MOVE(50)** moves the moving part 50 mm, or one station.

5-1-3-8 Example 6



The mechanical system consists of a ball screw. It uses a servo motor with a 17-bit absolute encoder. The mechanical gear ratio of the gearbox is 1:3. The screw pitch of the ball screw is 10mm per revolution. The total travel distance of the ball screw is 540 mm. The mechanical measurement units must be mm. With the same procedure as in example 1, we have:

$$\frac{Pn202}{Pn203}\,UNITS = \frac{2^{17}\cdot encoder\_counts}{1\cdot motor\_revolution}\,\frac{3\cdot motor\_revolution}{1\cdot ballscrew\_revolution}\,\frac{1\cdot ballscrew\_revolution}{10mm} =$$

$$= \frac{2^{17}\cdot 3}{10}\,\frac{encoder\_counts}{mm}$$

Therefore:

$$\frac{Pn202}{Pn203}\,UNITS = 2^{17}\,\frac{3}{10} = 2^{17}\,\frac{3}{2\cdot 5} = 2^{16}\,\frac{3}{5}$$

One solution is:

$UNITS = 2^{16} = 65536$
Pn202 = 3
Pn203 = 5

The calculation of the multiturn limit setting parameter Pn205 is not needed in this case because the ball screw is a system with a fixed (limited) axis. It is enough to set this value large enough to have the overflow of the counter out of the effective position. Also, because of the axis is finite, it is not important to set the **REP_OPTION** parameter, because **REP_DIST** must be set large enough so it is outside of the maximum effective position (540 mm). One solution is: **REP_DIST** = 1000 and **REP_OPTION** = 0.
With these setting, executing **MOVE(17)** moves the ball screw 17 mm in forward direction.

5-1-4 Mapping Servo Driver inputs and outputs

The CJ1W-MCH72 has got a digital I/O space that consists of 256 digital inputs and 256 digital outputs.
The digital outputs range has four parts:

- Digital outputs 0 - 7.
  These outputs do not physically exist on the CJ1W-MCH72. If you write these outputs, nothing happens. If you read these outputs, they return 0.
- Digital outputs 8 - 15.
  These outputs physically exist on the CJ1W-MCH72. You can physically access them on the 28-pin screwless connector on the front side of the CJ1W-MCH72. If you write these outputs, they become active and give a 24 VDC signal. If you read these outputs, they return their current status. Use the command **OP** to write and read these outputs.
- Digital outputs 16 - 255.
  These outputs can be mapped to the PLC memory. If they are not mapped, they are software outputs only. They do not physically exist on the CJ1W-MCH72, but you can write them and read their correct status. You use these outputs mostly in BASIC programs to accomplish some control sequences that require outputs which do not need to be physical. Use the command **OP** to write and read these outputs.

All outputs are unique to the controller. They are not accessed per axis.

The digital input range has three parts:

- Digital inputs 0 - 15.
  These inputs physically exist on the CJ1W-MCH72. You can physically access them on the 28-pin screwless connector on the front side of the CJ1W-MCH72. These inputs are active (ON) when a 24 VDC signal is applied to them. When you read them, they return their current status. Use the command **IN** to read these inputs.
- Digital inputs 16 - 255.
  These inputs can be mapped to the PLC memory. If they are not mapped, they are software inputs only. They do not physically exist on the CJ1W-MCH72, but you can read them. You use them mostly in BASIC programs to accomplish some control sequences that require inputs which do not need to be physical. Use the command **IN** to read these inputs.

All inputs are unique to the controller. They are not accessed per axis.

5-1-4-1 MECHATROLINK-II Servo Drivers inputs in the CJ1W-MCH72 I/O space

With the BASIC command **IN**, you can access the physically present inputs in a BASIC program. These inputs can be built in the controller or connected via the MECHATROLINK-II bus.
Servo Drivers can have additional inputs that are located on their I/O connectors. These inputs can be used as forward and/or reverse limit switches or origin switches. They are mapped into the CJ1W-MCH72 I/O space. Thus, they can be accessed from BASIC programs. The CJ1W-MCH72 only supports this for Servo Drivers connected to the CJ1W-MCH72 system via the MECHATROLINK-II bus. It is not supported for Flexible Axis Servo Drivers.

CJ1W-MCH72 input	Servo Driver input signal			Description
	Sigma-II	Sigma-V	Junma	
16	P_OT	P_OT	P_OT	Forward limit switch
17	N_OT	N_OT	N_OT	Reverse limit switch

CJ1W-MCH72 input	Servo Driver input signal			Description
	Sigma-II	Sigma-V	Junma	
18	DEC	DEC	/DEC	Zero point return deceleration
19	PA	PA	Not used	Encoder A phase signal
20	PB	PB	Not used	Encoder B phase signal
21	PC	PC	Not used	Encoder C phase signal
22	EXT1	EXT1	/EXT1	First external latch signal
23	EXT2	EXT2	Not used	Second external latch signal
24	EXT3	EXT3	Not used	Third external latch signal
25	BRK	BRK	/BRK	Brake output
26	Reserved	HBB	E-STP	Emergency stop switch
27	Reserved	Reserved	Not used	
28	IO12	IO12	Not used	Not used
29	IO13	IO13	Not used	Not used
30	IO14	IO14	Not used	Not used
31	IO15	IO15	Not used	Not used

The inputs in the table above are located on the CN1 I/O connector of the respective Servo Driver. The pin arrangement of this connector is different for the respective Servo Drivers. For the Sigma-II and Sigma-V Servo Drivers, the input signals P_OT, N_OT, DEC, EXT1, EXT2, EXT3, BRK, IO12, IO13, IO14 and IO15 can be mapped to pins of the CN1 I/O connector. To do this, you must set the appropriate parameter of the Servo Driver. The table below shows the possible settings and parameter values.

Input signal - Parameter name	Parameter setting	CN1 pin number	
		Sigma-II	Sigma-V
P_OT (active high) - Pn50A.3 N_OT (active high) - Pn50B.0 DEC (active high) - Pn511.0	0	40 (SI0)	13 (SI0)
	1	41 (SI1)	7 (SI1)
	2	42 (SI2)	8 (SI2)
	3	43 (SI3)	9 (SI3)
	4	44 (SI4)	10 (SI4)
	5	45 (SI5)	11 (SI5)
	6	46 (SI6)	12 (SI6)
	7	Always ON	
	8	Always OFF	

Input signal - Parameter name	Parameter setting	CN1 pin number	
		Sigma-II	Sigma-V
/P_OT (active low) - Pn50A.3 /N_OT (active low) - Pn50B.0 /DEC (active low) - Pn511.0	9	40 (SI0)	13 (SI0)
	A	41 (SI1)	7 (SI1)
	B	42 (SI2)	8 (SI2)
	C	43 (SI3)	9 (SI3)
	D	44 (SI4)	10 (SI4)
	E	45 (SI5)	11 (SI5)
	F	46 (SI6)	12 (SI6)
/EXT1 (active low) - Pn511.1 /EXT2 (active low) - Pn511.2 /EXT3 (active low) - Pn511.3	0-3	Always OFF	
	4	44 (SI4)	10 (SI4)
	5	45 (SI5)	11 (SI5)
	6	46 (SI6)	12 (SI6)
	7	Always ON	
	8, 9-C	Always OFF	
EXT1 (active high) - Pn511.1 EXT2 (active high) - Pn511.2 EXT3 (active high) - Pn511.3	D	44 (SI4)	10 (SI4)
	E	45 (SI5)	11 (SI5)
	F	46 (SI6)	12 (SI6)
/BRK (active low) - Pn50F.2	0	Always OFF	
	1	25	1
	2	27	23
	3	29	25
IO12 - Pn81E.0 IO13 - Pn81E.1 IO14 - Pn81E.2 IO15 - Pn81E.3	0	Always OFF	
	1	40 (SI0)	13 (SI0)
	2	41 (SI1)	7 (SI1)
	3	42 (SI2)	8 (SI2)
	4	43 (SI3)	9 (SI3)
	5	44 (SI4)	10 (SI4)
	6	45 (SI5)	11 (SI5)
	7	46 (SI6)	12 (SI6)

For the Junma Servo Driver, all input signals are mapped to a fixed location on the CN1 I/O connector. The table below shows the input signals and pin numbers.

Input signal	CN1 pin number
P_OT (active high)	4
N_OT (active high)	3
DEC (active low)	1
EXT1 (active low)	2
BRK (active low)	13
E-STP (active high)	6

For more information on the CN1 I/O connector pins on the Servo Drivers, refer to the Sigma-II Servo Driver manual, the Sigma-V Servo Driver manual, and the Junma series Servo Driver manual.

Servo Driver inputs that are mapped into the CJ1W-MCH72 I/O space like this are accessed within the program per axis and cannot be accessed in the usual way with the **IN** command. The only way you can use these inputs in the program is to assign them to the axis parameters **DATUM_IN**, **FHOLD_IN**, **FWD_IN** and **REV_IN**. The inputs of the axis Servo Driver are used, depending on the axis of which the parameters are set.

Example: We have a Sigma-II and a Junma driver assigned to controller axes 0 and 3. For the Sigma-II driver, we want to use input signal EXT1 (mapped to CN1-44 if Pn511.2 is set to 4) to serve as reverse limit input for axis 0. For the Junma driver, we want to use input signal EXT1 (CN1-2) as reverse limit for axis 3. We can do this with these commands:

REV_IN AXIS(0) = 22
REV_IN AXIS(3) = 22

Note that even though **REV_IN** parameters for both axes have the same value, the real inputs used are not the same. For axis 0 the input on CN1-44 of the Sigma-II driver (assigned to axis 0) is used, but for axis 3 the input on CN1-41 of the Junma driver (assigned to axis 3) is used. Therefore we say that those inputs are accessed per axis, they are not unique for the whole controller. In general, these two inputs have a different status at the same time. Also note that neither of these two inputs can be accessed using the command **IN**. For example the command **IN(22)** returns the status of controller software input 22 (unique for all axes), which has a different status than Servo Driver inputs mapped to the same number. However, the command **INVERT_IN(22)** inverts the status of input 22 read by the controller. It affects not only the unique software input 22, which is accessible with the **IN** command, but all axis-specific inputs 22, which in this example are the EXT1 inputs of the connected Servo Drivers.

Note If a forward limit, reverse limit and origin input signal are used for an axis, it is strongly recommended to use the following settings for the axis:

```
BASE(axis_number)
DAT_IN=18
' /DEC input in the corresponding Servo Driver is
' assigned

FWD_IN=16
INVERT_IN(16,ON)
' P_OT input in the corresponding Servo Driver is
‘ assigned. It is necessary to invert the signal
' because a Normally Closed input is expected.

REV_IN=17
INVERT_IN(17,ON)
' N_OT input in the corresponding Servo Driver is
‘ assigned. It is necessary to invert the signal
' because a Normally Closed input is expected
```

Also note that **INVERT_IN** inverts the selected input in all axes.

5-1-5 Origin search

The origin search or homing functionality is often seen as a particular sequence of movements of an axis at the start-up phase of the machine. This sequence is done automatically in most cases, without the input from the operator of the machine. In general, an origin search procedure couples a position to a specific axis. It depends on the encoders used (absolute or relative), on the system used (linear or circular), and on the mechanical construction of the machine. Absolute encoders do not need a movement during the origin search procedure, because the exact positions are transferred directly to the system. For other encoder types, a movement is necessary, since there is no knowledge of the exact position within the system. Basically, this movement is at low speed in some direction until a certain measuring point is reached. Such a measuring point can be scanned from both directions to increase the precision.
At startup, the current positions of the axes using incremental encoders are 0. Because these positions do not match with the mechanical 0 of the machine, it is necessary to execute the homing sequence. If an absolute encoder is used, the absolute position is read at startup from the encoder and homing is not necessary. In this case, a startup sequence must be executed one time during the machine commissioning.
In practice there are several different origin search sequences. They are different in these areas:

- The means used to detect limit positions of the moving part (sensors, switches, etc.)
- Origin (home) position or reference.
- Possible positions of the moving part related to limit positions and origin position.

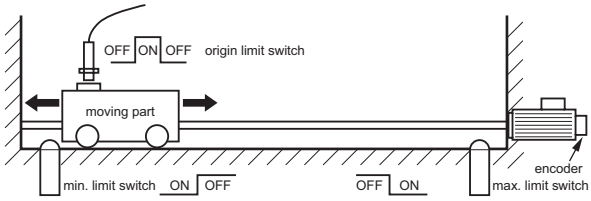
The CJ1W-MCH72 includes some pre-defined basic homing sequences:

- **DATUM(0)**
  This is not really an origin search. This command sets **DPOS=MPOS** and cancels the axis errors.
- **DATUM(1)**
  This does an origin search in forward direction using the Z mark of an encoder as homing switch.
- **DATUM(2)**
  Does an origin search in reverse direction using the Z mark of an encoder as homing switch.
- **DATUM(3)**
  Does an origin search in forward direction using the input selected in **DATUM_IN** as homing switch.
- **DATUM(4)**
  Does an origin search in reverse direction using the input selected in **DATUM_IN** as homing switch.
- **DATUM(5)**
  Does an origin search in forward direction using the input selected in **DATUM_IN** as homing switch and searches the next Z mark of an encoder.
- **DATUM(6)**
  Does an origin search in reverse direction using the input selected in **DATUM_IN** as homing switch and searches the next Z mark of an encoder.

For more details on these pre-defined homing sequences, see section 4-2-65.
In some situations, more complex homing sequences are required:

- Absolute switch origin search plus limit switches.
- Origin search against limit switches.
- Origin search against hardware parts blocking movement.

- Origin search using encoder reference pulse "Zero Mark".
- Static origin search, forcing a position from a user reference.
- Static origin search, forcing a position from an absolute encoder.



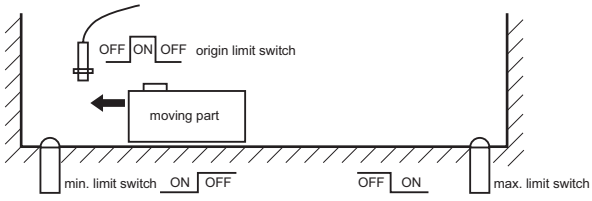
The figure shows a general origin search scenario. This simple origin search sequence has 3 steps:

1 Search for a signal.
2 Search for another signal.
3 Move the axis to a predefined position.

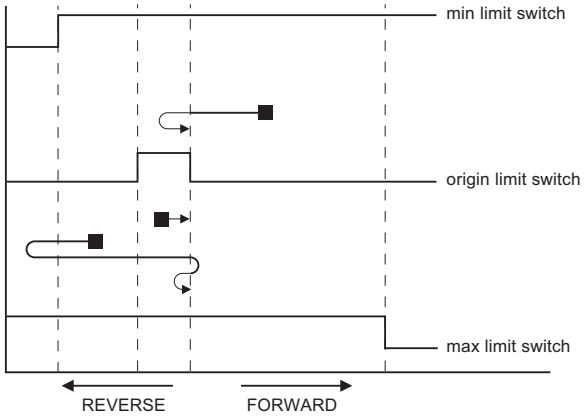
Note For safety reasons, limit switches are normally closed. For this reason, in this figure and in the following figures in this section, the low signal level is indicated as ON, and the high signal level is indicated as OFF.

It is important to note that, before any homing procedure is executed, it is necessary to set the axis parameters **UNITS**, **REP_DIST** and **REP_OPTION**, and Servo Driver parameters Pn202, Pn203 and Pn205 properly and in accordance with the mechanical system and desired measurement units used in programming. Those parameters have influence to the origin search, especially if an absolute encoder is used. For more information on setting these parameters, see section 5-1-2.

5-1-5-1 Absolute switch origin search plus limit switches



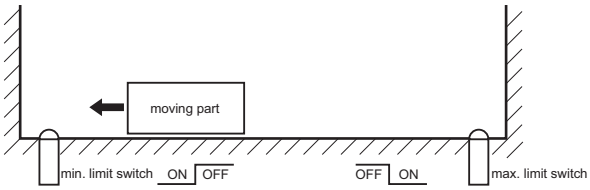
The origin search function is performed by searching for an external limit switch that is positioned absolutely and the position of which defines the origin position. The example for this homing procedure is shown in the figure.



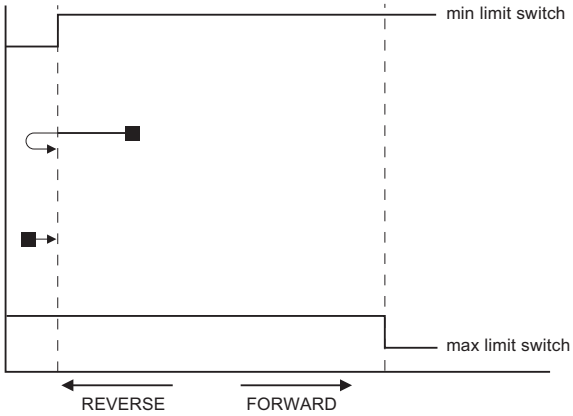
The figure shows the possible scenarios for absolute origin search plus limit switches. These scenarios depend on the position of the moving part when the power comes on.
The program example that does this origin search sequence is given below.

```
'Absolute origin switch: IN0
'Left limit switch: IN1
'Right limit switch: IN2
BASE(0)
DATUM_IN=0
FW_IN=2
RV_IN=1
SERVO=ON
WDOG=ON
DATUM(4)
WA(1)
WAIT UNTIL MTYPE=0 OR IN(1)=OFF
IF IN(1)=ON
    FORWARD
    WAIT UNTIL IN(0)=ON
    WAIT UNTIL IN(0)=OFF
    CANCEL
    DATUM(4)
    WA(1)
    WAIT IDLE
ENDIF
```

5-1-5-2 Origin search against limit switches



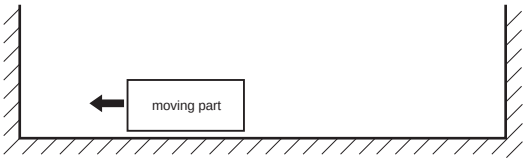
This origin search function is performed by searching for an external sensor using limit switches only. The example for this homing procedure is shown in the figure.



The possible scenarios for origin search against limit switches, depending on the position of the moving part on power on, are shown in the figure.
The program example that does this origin search sequence is given below.

```
'Origin and left limit switch: IN0
'Right limit switch: IN1
BASE(0)
DATUM_IN=0
SERVO=ON
WDOG=ON
DATUM(4)
WA(1)
WAIT IDLE
```

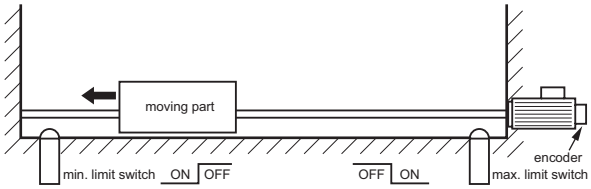
5-1-5-3 Origin search against hardware parts blocking movement



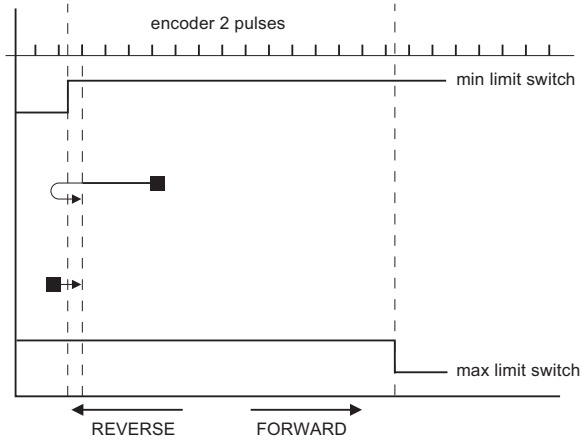
This origin search procedure performs origin search against a physical object and mechanically blocks the movement. There are no limit switches, no absolute position switch and no reference pulses. The origin position is detected by detecting a particular amount of torque against the blocking objects. An adequate torque limit is required in order not to damage the mechanics during the origin search process. The example for this homing procedure is shown in the figure.
The program example that does this origin search sequence is given below.

```
BASE(0)
DRIVE_CONTROL=11 'Monitor torque with DRIVE_MONITOR
SERVO=ON
WDOG=ON
SPEED=CREEP
REVERSE
WA(1)
WAIT UNTIL DRIVE_MONITOR < -100
    'Wait for particular amount of applied torque
CANCEL
DEFPOS(0)
MOVEABS(10) 'This is necessary, otherwise the position
            'is kept pushing the hardware limit of the
            'machine and the motor trips by overload
```

5-1-5-4 Origin search using encoder reference pulse "Zero Mark"



This origin search procedure performs origin search by searching for the "Zero Mark" signal of the encoder. This signal is also known as "marker" or "reference pulse". It appears one time per full encoder revolution. The example for this homing procedure is shown in the figure.



The possible scenarios for origin search using encoder reference pulse "Zero Mark", depending on the position of the moving part on power on, are shown in the figure.
The program example that does this origin search sequence is given below.

```
'Origin and left limit switch: IN0
'Right limit switch: IN1
REV_IN=-1
BASE(0)
DATUM_IN=0
SERVO=ON
WDOG=ON
DATUM(6)
WA(1)
WAIT IDLE
```

5-1-5-5 Static origin search, forcing a position from a user reference

This origin search procedure performs a static origin search by directly forcing an actual position. It does not perform any physical move.

```
DATUM(0)
```

5-1-5-6 Static origin search, forcing a position from an absolute encoder

This origin search procedure sets the actual position to the position of an absolute encoder. It does not perform any physical move. It is only possible with an axis with an absolute encoder in a control loop.

5-1-6 Registration

Registration, also called 'latch' or 'print registration', is about real-time storing of the position of an axis when an external input is activated. The information that is registered, i.e. stored, is processed later, not in real time, by the application program.
Registration is different from processing an interrupt input or signal. With registration, no event is generated when the registration input is activated. Also, the normal execution of the application program is not disturbed or

interrupted. Only the position of an axis is stored. This information can be used, like other parameters or values, in a program. The registration information is available to a program immediately after the registration.

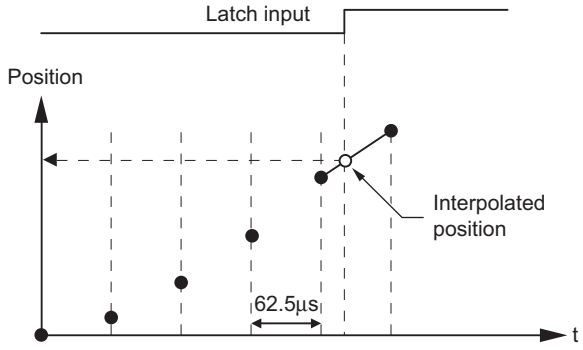
The advantage of registration is that it is done very quickly. Therefore, the axis position that is stored is very accurate. To achieve this speed and accuracy, registration is implemented with hardware, and the registration input must be on the same board as the encoder input that provides information on the axis position.
Capturing and storing the axis position is done in real time by the hardware. Processing this information is done not in real time by the application program.

5-1-6-1 The REGIST axis command

In Trajexia, you do a registration with the **REGIST** axis command. This command takes one argument. This argument determines which external input is registered, whether the registration is executed on the rising edge or on the falling edge of the input signal, whether the windowing function is used, and other options. For more information on the **REGIST** command, refer to section 4-2-198.
The registration differs for different axes depending on their connection to the system. If an axis is connected via the MECHATROLINK-II bus, the registration is done in the Servo Driver hardware. If an axis is connected via the Encoder Interface, the registration is done in the hardware of the CJ1W-MCH72.
The different registrations are described below.

5-1-6-2 Registration in the Sigma-II and Sigma-V Servo Driver

Registration in the Sigma-II and Sigma-V Servo Driver occurs when an axis assigned to this Servo Driver is connected to the Trajexia system via the MECHATROLINK-II bus. There are three registration inputs on these Servo Drivers, but only one hardware latch, so only one input can be used at a time. For Sigma-II Servo Drivers the physical inputs are in pins CN1-44, CN1-45 and CN1-46 on the 50-pins CN1 connector. For Sigma-V Servo Drivers the physical inputs are in pins CN1-10, CN1-11 and CN1-12 on the 26-pins CN1 connector. Trajexia uses logical inputs EXT1, EXT2 and EXT3 to associate the physical inputs to logical ones. This association is done by setting the parameter Pn511 of the Servo Driver. For more information on setting this association and Pn511 parameter, refer to section 4-2-198, table 1. The input used for registration is determined by the argument of the **REGIST** command.



The delay in the capture in the Sigma-II Servo Driver is about 3 μs. As the encoder information is refreshed every 62.5 μs, it is necessary to make interpolation to obtain the right captured position value (see the picture). Since the motor speed cannot change much during 62.5 μs, the resulting accuracy is very high.
The delays in transmission of the information are:

- Delay in triggering the registration: 0.625 ms to 4 ms.
- Delay in receiving the registration: 3.5 ms.
- Delay in capturing the registration: 3 μs.

It is also possible to use the encoder Z-mark to register an axis position. This is also done with the argument of the **REGIST** command.

5-1-6-3 Registration in the Junma Servo Driver

Registration in the Junma Servo Driver is the same as registration in the Sigma-II Servo Driver, with one difference: There is only one physical input and one logical latch too, so no settings of Servo Driver parameters are necessary. The physical input is associated to logical latch EXT1, and only the rising signal edge can be used for registration.

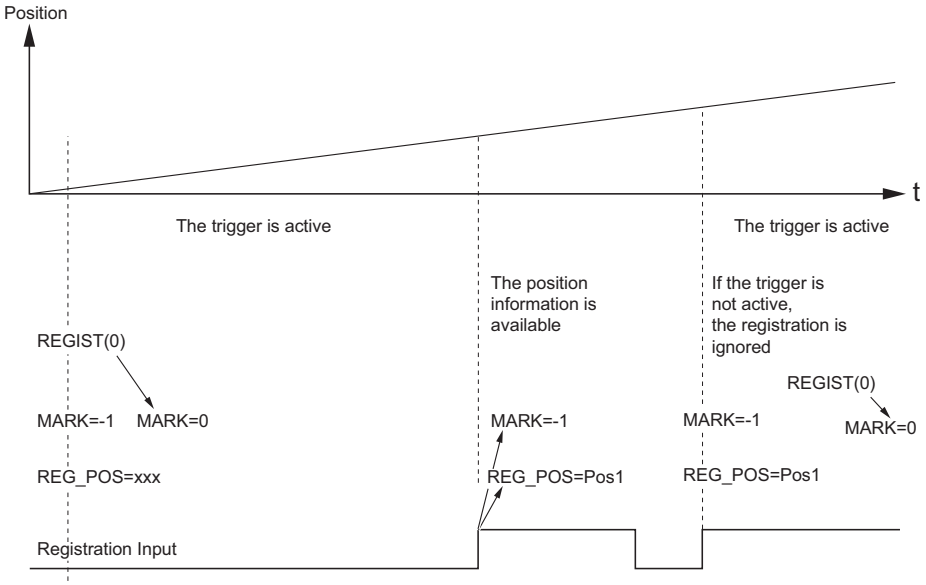
5-1-6-4 Registration in the Encoder Interface

The CJ1W-MCH72 has three physical registration inputs, and two latch circuits, which can be used independently. Therefore two independent registration inputs can be used at the same time. For more information on how to use both registration inputs of the CJ1W-MCH72 at the same time, refer to sections 4-2-150, 4-2-151, 4-2-196, 4-2-197 and 4-2-198.
The delay in the capture is 0.5 μs. Because the encoder position is read continuously from the line-drive encoder input, interpolation is not necessary. The delay for the transmission of the captured information is just one **SERVO_PERIOD** cycle.

5-1-6-5 Using registration in application programs

There is one axis command (**REGIST**), and two axis parameters (**MARK** and **REG_POS**). With these commands and parameter, you can control and use the registration functionality in BASIC programs.

- **REGIST** captures the axis position when a registration signal is detected. The available settings depend on the axis type. Refer to section 4-2-198.
- **MARK** is a flag that signals whether the position has been captured or not. For the second registration input of the Encoder Interface, the parameter **MARKB** is also available. For more information, refer to sections 4-2-150 and 4-2-151.
- **REG_POS** holds the captured axis position. Only if the **MARK** flag signals that the position was captured successfully, you can regard the **REG_POS** value as valid. For the second registration input of the Encoder Interface, the parameter **REG_POSB** is also available. For more information, refer to sections 4-2-196 and 4-2-197.



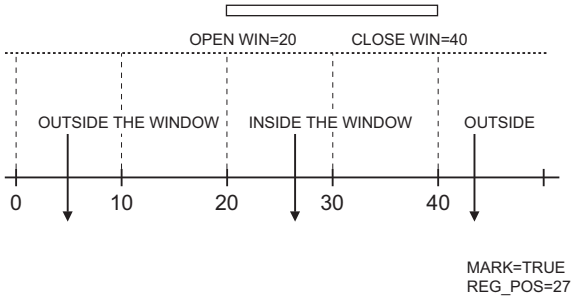
The picture gives the sequence of executing the commands and the registrations of the sample program below.

```
BASE(N)
REGIST(0)
WAIT UNTIL MARK=0
loop:
    WAIT UNTIL MARK=-1
    PRINT "Position captured in: "; REG_POS
    REGIST(0)
    WAIT UNTIL MARK=0
GOTO loop
```

5-1-6-6 Registration and windowing function

The windowing function enables for registration to occur only within a specified range of axis positions. This function is selected by giving the right value as an argument for the **REGIST** command. The windowing function is controlled by two axis parameters, **OPEN_WIN** and **CLOSE_WIN**. For more information on **REGIST**, **OPEN_WIN** and **CLOSE_WIN**, refer to sections 4-2-198, 4-2-176 and 4-2-51.

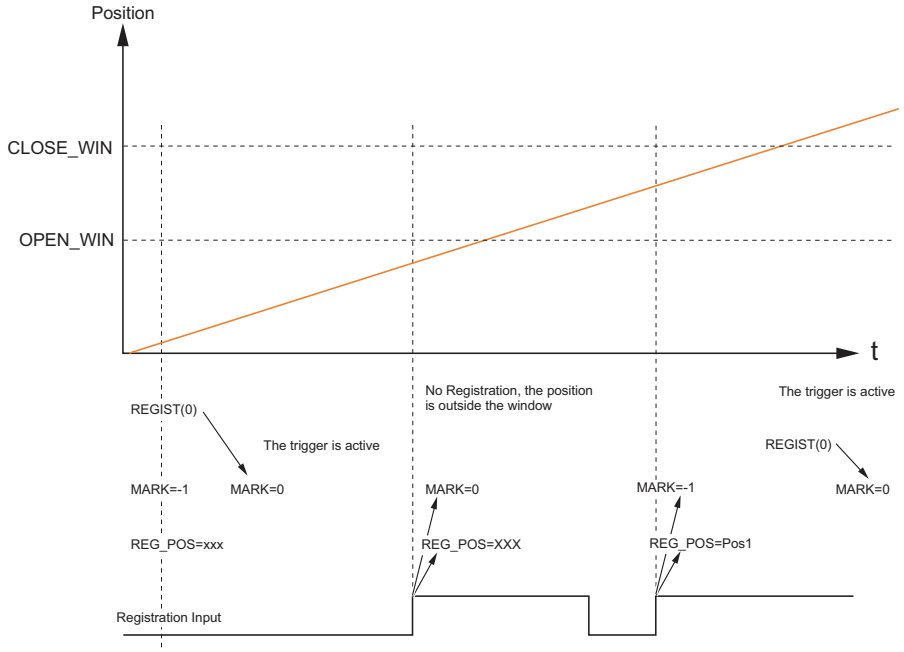
There are two types of windowing:



- Inclusive windowing allows the registration to occur only within the specified window of axis positions. With this windowing function, registration events are ignored if the axis measured position is less than the **OPEN_WIN** axis parameter or greater than the **CLOSE_WIN** parameter.
- Exclusive windowing allows the registration to occur only outside the specified window of axis positions. With this windowing function, the registration events are ignored if the axis measured position is greater than the **OPEN_WIN** axis parameter or less than the **CLOSE_WIN** parameter.

When the windowing function is used, the internal process is as follows:

1 **REGIST** + window is executed in the program.
2 **MARK** = 0 and the latch is triggered.
3 The position is captured and transmitted to the Trajexia processor.
4 Is the captured position inside the inclusive window or outside the exclusive window?
   - If yes, **MARK** = -1 and **REG_POS** is updated.
   - If not, return to point 2 (trigger the latch again transparently to the user).

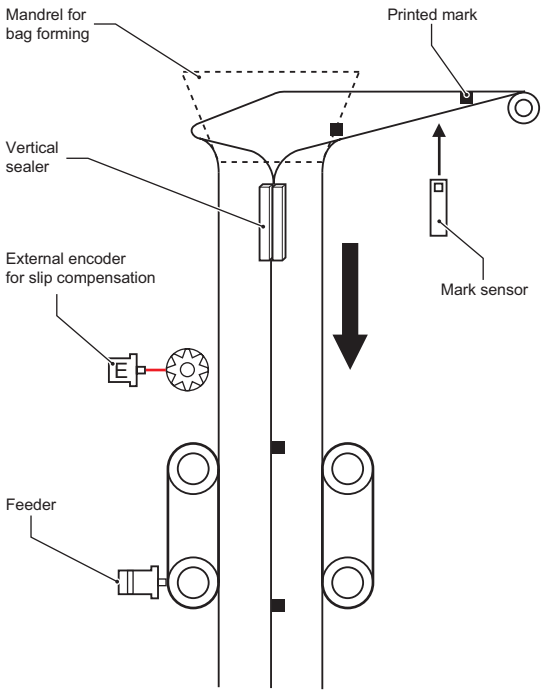


The figure shows the sequence of execution of the above commands and the occurrence of registration events when you use inclusive windowing.
There are delays between these events:

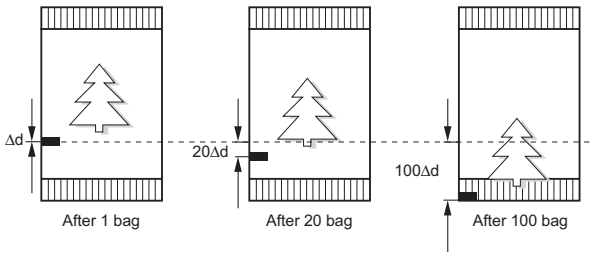
- Trajexia receives the latch.
- Trajexia decides to trigger the latch again.
- The latch is triggered.

Because of these delays, there is an uncertainty in the edges of the window when marks may be detected near the edges. This is more notable for axes connected to the system via the MECHATROLINK-II bus due to bus delays. To compensate for these delays, a user must set the window margins large enough.

5-1-6-7 Example: Correcting the position of an axis

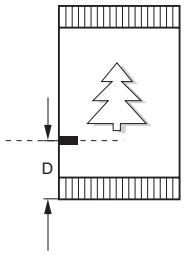


The picture shows the vertical fill and seal machine for packaging products into bags. The bag material comes from a plastic film coil that is unwinded, then it is shaped into the tube by a mechanical mandrel and at the same time the tube is sealed vertically. The feeder movement is intermittent and the feed length corresponds with the bag length. Once the bag is fed, the horizontal sealer closes the bag, so it can be filled with the product. After that, the process starts again, feeding the new bag.

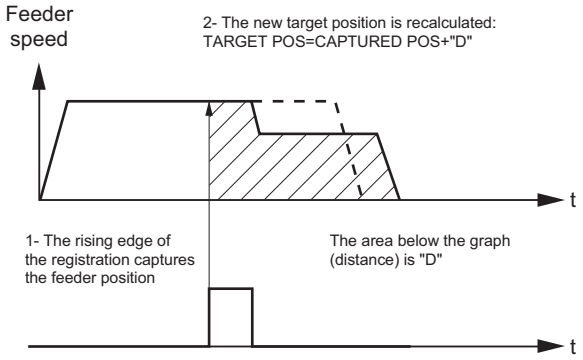


The feeder can work in two modes: without registration mark; and with registration mark. Working without the registration mark is a simple point-to-point incremental movement. In this case, there is no guarantee that the feeder moves exactly the same distance as the design pattern. For example, suppose the bag length that needs to be fed is 200 mm, but the real pattern is 200.1 mm. With simple point-to-point incremental movement without

correction, an error of 0.1 mm per bag is accumulated. With a small number of bags the difference is not visible, but after 500 bags the error is 50 mm, which is a 25% of the bag length.



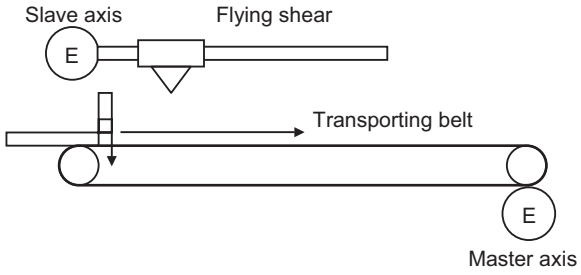
When working with registration marks, the motion controller executes an incremental movement to a certain position. If during the positioning the registration mark is detected, the target position is changed on the fly in order to finish the movement at a defined position after the registration mark. Therefore, the same distance in respect to the registration mark is always guaranteed.


Feeder speed
2- The new target position is recalculated:
TARGET POS=CAPTURED POS+"D"
t
1- The rising edge of the registration captures the feeder position
The area below the graph (distance) is "D"
t

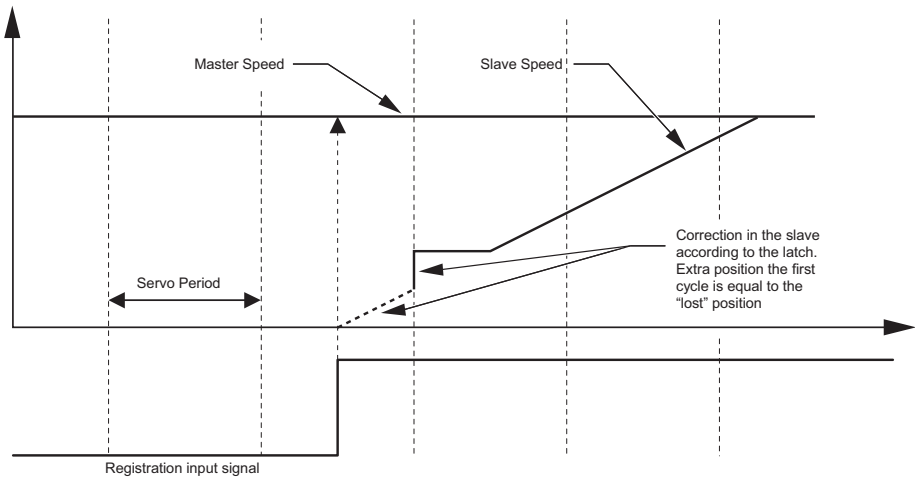
The motion profile and its modification due to the registration mark are shown in .
The BASIC program for this example is:

```
DEFPOS(0)
REGIST(3)        'Trigger the mark registration
MOVE(bag_length) 'Move to the theoretical distance
WA(1)
WAIT UNTIL MARK OR MTYPE=0
IF MARK THEN
    end_position=REG_POS+distance_after_mark
    MOVEMODIFY(end_position)
    'Correct the distance according to the mark
ENDIF
```

5-1-6-8 Example: Starting a slave axis in precise position of a master axis



The picture shows a flying shear cutting the "head" of wood tables. When the wood comes, the edge of the wood is detected by the photocell and, at the exact moment, the movement of the flying shear starts to be synchronized with the right position on the wood.
If the movement is started by the program, upon detecting a signal from the photocell, there is always at least one **SERVO_PERIOD** of time of uncertainty. Instead, the movement is started using the **MOVELINK** command with **link_option**=1, which means that the link to the master axis starts when the registration event occurs on link (master) axis.
The corresponding program sequence is:
REGIST(2) AXIS(master)
MOVELINK(dst,lnk_dst,lnk_acc,lnk_dec,master,1) AXIS(slave)
For more information on the **MOVELINK** command and the **link_option** argument, refer to section 4-2-160.


Master Speed
Slave Speed
Correction in the slave according to the latch. Extra position the first cycle is equal to the "lost" position
Servo Period
Registration input signal

The picture shows how the position of the slave axis is corrected using the registration event on the master axis to start the movement of the slave axis. The influence of **SERVO_PERIOD** and the fact that the registration event can happen at any time inside the **SERVO_PERIOD** is completely eliminated.

5-1-7 Tracing and monitoring

5-1-7-1 Oscilloscope functionality in Trajexia Studio

The software oscilloscope is a standard part of Trajexia Studio. The oscilloscope can be used to trace and graphically represent axis and system parameters. This can help you with development, commissioning and troubleshooting of the motion system. For more information on the software oscilloscope and its features and capabilities, refer to the Trajexia Studio manual.

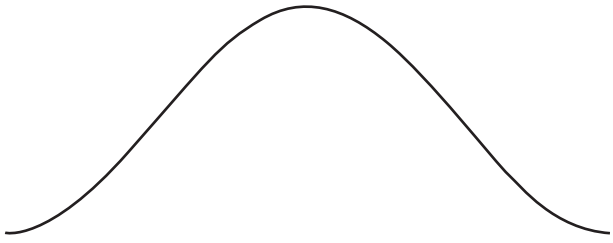
You can trigger the oscilloscope to start tracing given axis and system parameters in two ways: manually or by a program. Triggering manually is done using the data trace. The parameters are stored in the Table memory of the controller. The range of the Table memory where the parameters are stored can be set in the **Memory Manager** of the device configuration (see the Trajexia Studio manual). With manual triggering, the user can see the changes of axis and system parameters in real time, as the system runs. A change in parameter values is graphically represented as soon as the change happens. The limitation of manual triggering is that it requires user interaction, which means that the start of tracing is not synchronized with the movement that is analyzed. Also, with manual triggering the tracing range is limited to 200 samples per channel.

5-1-7-2 Using the oscilloscope

The alternative, triggering by a program, does not have the limitations of manual triggering of the tracing. Triggering by a program stores the axis and system parameters in the memory of the CJ1W-MCH72. Later, the parameters are given to the oscilloscope for graphical representation. The axis and system parameters are stored in the Table memory. The memory range used is defined by the parameters of the **SCOPE** command. When the parameters are in the Table memory, the oscilloscope can be configured to show a range of Table memory locations instead of axis and system parameters. The exact moment when the tracing is started can be exactly determined because it is controlled by the **TRIGGER** command. This means the start of tracing is synchronized with the movement.There is no limitation of 200 samples per channel, the oscilloscope shows as many samples (Table entries) as configured.

5-1-7-3 Example

This section gives you a practical example on the use of the **SCOPE** and **TRIGGER** commands, and how to use them in combination with the oscilloscope to monitor axis parameters and troubleshoot the system. For more information on the **SCOPE** and **TRIGGER** commands, refer to sections 4-2-216 and 4-2-243.



Suppose the motion system consists of two axis, **AXIS(0)** and **AXIS(1)**. **AXIS(0)** is the master axis. It makes a simple forward movement. **AXIS(1)** is the slave axis. It must follow the master axis in accordance to cosine rule:

$$x_1 = end\_pos \cdot \frac{1}{2}\left(1 - \cos\left(\frac{2\pi \cdot x_0}{999}\right)\right)$$

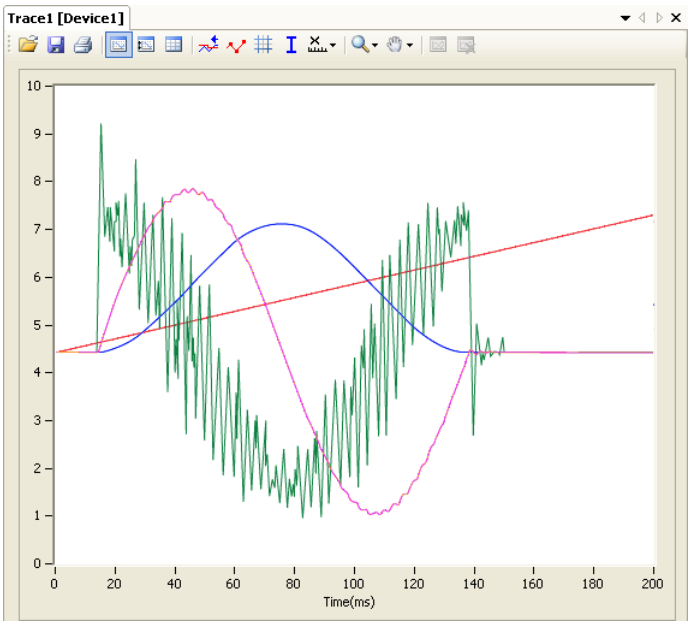
where x_0 is the position of the master **AXIS(0)**, and x_1 is the position of the slave **AXIS(1)**. You can link the two axis with the **CAMBOX** command. For more details, refer to section 4-2-44. Suppose furthermore that the parameter **end_pos** is not constant, but it can change due to different conditions of the motion system. The part of the program that creates the CAM table is:

```
'Initial CAM values
VR(end_pos)=15
current_end_pos=VR(end_pos)
FOR i=0 TO 999
    TABLE(i, VR(end_pos)*(1-COS(2*PI*i/999))/2)
NEXT i
...
loop:
IF VR(end_pos)<>current_end_pos THEN
'Recalculate the CAM Table
    FOR i=0 TO 999
        TABLE(i, VR(end_pos)*(1-COS(2*PI*i/999))/2)
    NEXT i
    current_end_pos=VR(end_pos)
ENDIF
...
GOTO loop
```

The **VR(end_pos)** value can be changed from some other program or externally from another controller using FINS messaging. In this case, the **CAM** table must be recalculated.
The creation of the CAM table is complete. The initialization of the desired axis and system parameters for tracing is:

```
'Initializations
FOR i=0 TO 1
    BASE(i)
    ATYPE=40
    UNITS=8192
    REP_DIST=20
    REP_OPTION=1
    FE_LIMIT=1
    DRIVE_CONTROL=11
    SPEED=8
    ACCEL=50
    DECEL=50
    DEFPOS(0)
    SERVO=ON
    CANCEL
NEXT i
WDOG=ON
BASE(1)
'Scope settings:
```

```
'1 sample each 2 servo cycles
'Information stored in TABLE(1000) to TABLE(4999)
'Because we capture 4 channels, we have 1000 samples per
channel.
'MPOS AXIS(0) is stored in TABLE(1000) to TABLE(1999)
'DPOS AXIS(1) is stored in TABLE(2000) to TABLE(2999)
'Torque reference for AXIS(1) is stored in
'TABLE(3000) to TABLE(3999)
'MSPEED AXIS(1) is stored in TABLE(4000) to TABLE(4999)
'The capture covers 1000 samples * 2ms / sample = 2seconds
SCOPE(ON,2,1000,4999,MPOS
AXIS(0),DPOS,DRIVE_MONITOR,MSPEED)
FORWARD AXIS(0) 'Move the master axis forward
TRIGGER 'Start tracing and storing of parameters
WHILE NOT MOTION_ERROR
    'Cambox that will start in AXIS(0) position 1
    CAMBOX(0,999,UNITS,10,0,2,1)
    WAIT UNTIL MPOS AXIS(0)<1
    'The capture will start when the master axis is in
    'a position Between 0 and 1. Additional conditions
    'are:
    '- The previous capture has finished
    '(SCOPE_POS=1000)
    '- We have the permission (VR(activate_trigger)=ON)
    IF SCOPE_POS=1000 AND VR(activate_trigger)=ON THEN
        TRIGGER
        PRINT "Triggered"
    ENDIF
    WAIT IDLE
WEND
HALT
```



The result is given in the figure.

In the example given above, the value of the **UNITS** parameter is set to encoder counts. The position of the master axis **MPOS AXIS(0)** is given in red. The position increases linearly, because the speed of the master axis is constant.
The demanded position of the slave axis **DPOS AXIS(1)** is given in blue. This graph is a cosine curve. It corresponds to the created CAM table.
The measured speed of the slave axis **MSPEED AXIS(1)** is given in yellow. This graph is a sinusoidal curve, because the speed is a derivative of the position, and the derivative of the cosine is the sine. At high speeds, there are some ripples.
The green graph is the torque of the motor for the slave axis set with **DRIVE_COMMAND=11** as a percentage of the nominal torque. The torque is proportional to the acceleration. Because the acceleration is a derivative of the speed and the speed is sinusoidal curve, the acceleration (and also the torque) is a cosine curve. There is one peak at the start and another peak at the stop because there is a discontinuity in the acceleration. There is also a high frequency oscillation in the torque curve, suggesting a resonance frequency that can be eliminated using the notch filter settings in the Sigma-II Servo Driver. The high frequency is reinforced, because it is also reflected in the speed curve. For more information on notch-filter settings, refer to the Sigma-II Servo Driver manual.

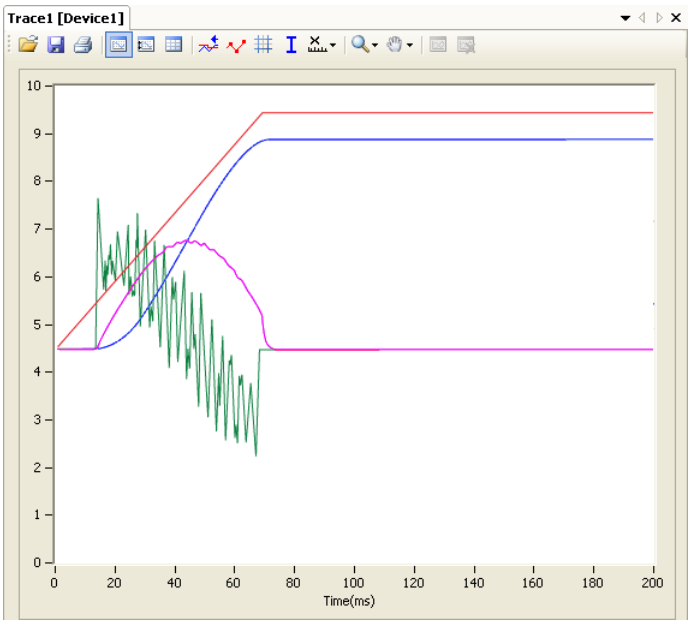
5-1-7-4 Troubleshooting with the oscilloscope

When the desired data is captured and recorded into the Table memory entries, you can use the oscilloscope to visualize this data. This can help you when you commission and troubleshoot the system. This section gives an example of how a bug, which is difficult to analyze, can be clearly explained and solved using the captured data and the oscilloscope.

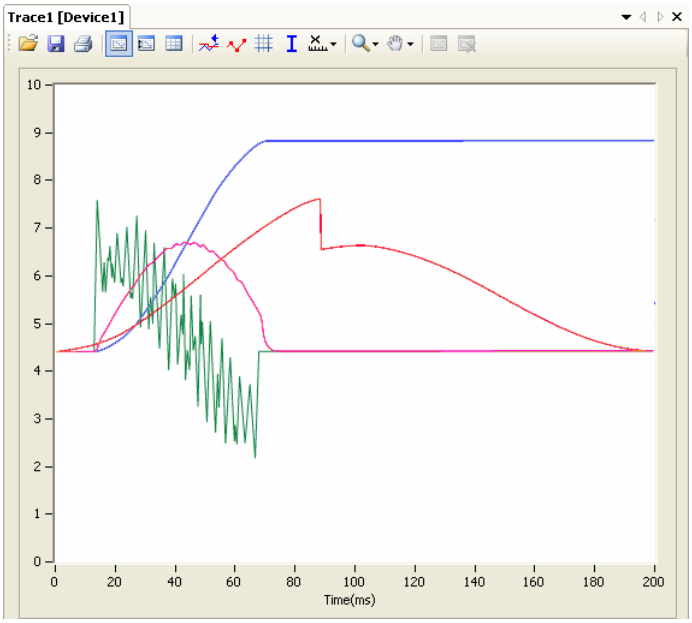
The parameter **end_pos**, which defines the values in the CAM table, depends on external conditions of the system. Therefore a program that runs in another task or even a controlling device using FINS communication, can change it while the main program that links two axis runs. Suppose that these changes in conditions, which result in a change of the **end_pos** parameter, happen most of the time when the axes are not linked, i.e. when the **CAMBOX** command is not executed. Suppose furthermore that very rarely the condition changes when the axes are linked. The change of the **end_pos** parameter triggers the recalculation of the CAM table while the **CAMBOX** command is executed. The consequence is that the part of the demanded position of the slave axis follows the profile before the change, and the other part follows the profile after the change. In the end this leads to a discontinuation of the profile, which causes an indefinite speed of the axis and ends up with this error: the WDOG goes off, and all axes stop.
The scenario above is hard to analyze when you do not know what happens. The only thing that the user sees is that the slave axis has an error once every few hours or even less often. But the oscilloscope can clearly show where the problem is. In order to be able to use the oscilloscope, all desired parameters must be captured at the time of an error. This can be achieved by arranging the application programs in a certain way. The good programming practice suggests to have a separate start-up program that is set to run automatically on power-up of the system and checks the integrity of the system, whether all the expected slaves are connected and initialized. For an example of a start-up program see section 5-1-1. It is recommended to let the start-up program, when it is finished, start only one program that takes care of the safety and

integrity of the application and execution of all other application programs. This program is usually referred to as a SHELL program. For more information on designing a SHELL program, see section 5-2-1.
Suppose that program is designed in a way the it contains a following fraction of code:

```
'When there is an error, we stop all programs. No new
'oscilloscope captures are done. And we have stored in
'the selected TABLES the last data trace in which the
'error has occurred. Therefore, we can recover this
'trace and analyze it.
loop:
    IF MOTION_ERROR<>0 THEN HALT
GOTO loop
```



This programming code causes all the programs and tracing to stop when an error happens on any axis. The data is already captured in the Table memory, and we can start using the oscilloscope to see the status of the desired parameters at the moment the error occurred.
The measured position of the master axis, given in red, does not seem to be the cause, because there is no discontinuity on it. We discard a mechanical problem as well, because the torque, given in green, has low values. An the moment of the problem the speed of the slave axis, given in yellow, was smooth and low, therefore this is no problem either.
The next step is to analyze the CAM table, to see which values were used for demanding the position of the slave axis. To do that, we change the data trace to show a block of values from Table(0) to Table(999) in red, because these entries are where the CAM table is created (see the part of the program that creates the CAM table above). The changed configuration is shown in the figure.

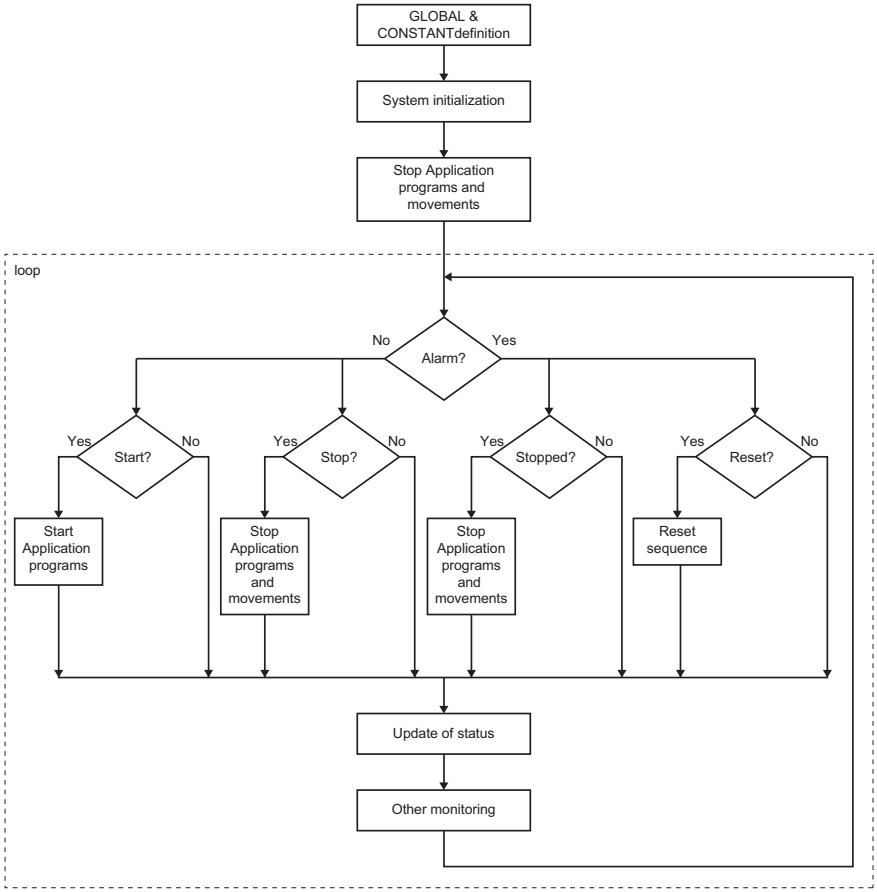


The result is given in the figure. The red graph clearly shows a discontinuity in the position values that the slave axis must follow. Because the speed is a derivative of the position, at the point of discontinuity of the position curve the speed gets a high value. (This value equals infinity in theory, in practice the value is just very big). This causes the error. The red graph shows where the root of the problem is. The amplitude of the cosine curve, and therefore the **end_pos** parameter, has been changed during the execution of the **CAMBOX** command. The solution is simple: A change of the **end_pos** parameter during **CAMBOX** execution must be prevented. To do this, either modify the programs in Trajexia, or in some other controller (if the parameter is changed outside of the scope of the application programs, for example by a FINS message).

Note The time base of the CAM TABLE points is not the same as the capture of the other signals. The discontinuity in the CAM (red graph) coincides in time with the interruption of the movement. To analyze this, check the position values individually with a spreadsheet program. To analyze the point values in detail, you can export the TABLE points to a spreadsheet program for a more complex analysis.

5-2 Practical examples

5-2-1 SHELL program



Good programming practice requires a good SHELL program. A SHELL program starts, stops and resets the application programs. The SHELL program is not necessary, but gives structure to the applications and makes the method to program the motion controller more effective.
Find below an example of a SHELL program. Make sure that you modify the program to the specific needs of the application. Check the correct operation before you rely on the safe operation of the program. This program is typically set to run at power-up at low priority.

5-2-1-1 Example

```
'=======================================================
=======================
'This SHELL program is an example that OMRON provide as
recommended. This
' program should be modified for the particular user
application.
'=======================================================
=======================

'Reserved symbol area for SHELL program handling. Do not
use these areas in your
' application programs:
```

```
'VR(900) - "status_word" reports about the status of the
system
'   =0 during initialization
'   =1 application stopped with no error
'   =2 errors in the system
'   =3 application running

'VR(901) - VR(status_bits) reports next status
'   Bit0    Alarm flag
'   Bit15   ML communication error with one slave

'VR(902) - "action" send messages to the upper controller
'   =0 during initialization
'   =1 Push RESET to restart
'   =2 Resetting
'   =3 System healthy

'VR(903) - VR(diag01) gives feedback of the MECHATROLINK
initialisation
'   Bit0    Could not get the ML slave number
'   Bit1    Slave number is uncorrect
'   Bit15   Detection OK

'VR(904) - VR(diag02) gives feedback of the MECHATROLINK
Slaves
'   Bitn    Slave n not detected

'VR(905) - VR(diag03) gives feedback forUnit detection
'   Bitn    Unit n detected

'VR(906) - VR(unit_detection) used in detecting units

'VR(907) - VR(signal_state) gives feedback on signal state

'VR(908) - "sys_error" system error detected

'VR(909) - "first_error" gives the axis number causing a
motion error

'VR(910,912 ... 940) - VR(servo_status+axis_n*2) stores
AXISSTATUS to report
'                     to upper controller

'VR(911,913 ... 941) - VR(servo_alarm+axis_n*2) stores the
alarm code of
'                     the servo

'Omron Auto Generated - Symbols
'Warning: Automated code section - any manual code changes
will be lost.

'Omron Auto Generated - Globals
GLOBAL "status_word",900
GLOBAL "action",902
GLOBAL "sys_error",908
```

```
GLOBAL "first_error",909

'Omron Auto Generated - Constants
CONSTANT "max_axis",15
CONSTANT "status_bits",901
CONSTANT "diag01",903
CONSTANT "diag02",904
CONSTANT "diag03",905
CONSTANT "unit_detection",906
CONSTANT "signal_state",907
CONSTANT "servo_status",910
CONSTANT "servo_alarm",911

'PLC Exchange Settings

'Omron Auto Generated - CAM TABLE

'Omron Auto Generated - Symbols End

'Omron Auto Generated - Local Variables
alarm_bit=0
i=0 'Servo Parameters
res=0 'Servo Parameters
res_act=0
res_ant=0
res_bit=0
run_act=0
run_ant=0
run_bit=0
stop_act=0
stop_ant=0
stop_bit=0
'Omron Auto Generated - Local Variables End

VR(signal_state) = 0

'At least the right system needs to be properly detected
once
GOSUB system_detection

'Stop all potential programs movements
GOSUB stop_all

status_word=1

loop:

    IF alarm_bit THEN
        action=1 ' Alarm, push RESET to restart

        IF status_word<>2 THEN
            PRINT "Stop with Alarm"
            GOSUB stop_all
            status_word=2 'Programs stopped with error
        ENDIF
```

```
            IF res_bit=1 THEN
                action=2 'Resetting
                PRINT "Resetting"
                GOSUB reset_all
                status_word=1 'Programs stopped NO error
            ENDIF
        ELSE
            action=3 'OK

            IF run_bit=1 THEN
                PRINT "Start application"
                GOSUB start_application
                status_word=3 'Application running
            ENDIF

            IF stop_bit=1 AND status_word=3 THEN
                PRINT "Stop by command"
                GOSUB stop_all
                status_word=1
            ENDIF
        ENDIF

        'Evaluates rising edge in RUN, STOP & RESET bits
        GOSUB sequence

         'Checks for alarms in the system and monitors the
system status
        GOSUB alarm_sequence

        'Upgrade values for showing in the HMI & PLC
        GOSUB monitoring

        'Reports and reset warnings in servodrive
        GOSUB warning_seq

GOTO loop

'-------------------------------------------------------
-----------------------

sequence:

'Define here your signals to STOP/START/RESET

'This example uses the following signals:
'Rising edge of bit 0 of VR(signal_state) as RUN signal
'Rising edge of bit 1 of VR(signal_state) as STOP signal
'Rising edge of bit 2 of VR(signal_state) as RESET signal

'RUN
run_ant=run_act
run_act=READ_BIT(0,signal_state)
run_bit=run_act AND NOT run_ant

'STOP
stop_ant=stop_act
```

```
stop_act=READ_BIT(1,signal_state)
stop_bit=stop_act AND NOT stop_ant

'RESET
res_ant=res_act
res_act=READ_BIT(2,signal_state)
res_bit=res_act AND NOT res_ant

RETURN

'-------------------------------------------------------
----------------------

alarm_sequence:

'Alarm notification
IF     SYSTEM_ERROR=0     AND     MOTION_ERROR=0     AND
READ_BIT(15,diag01)=1 THEN
    alarm_bit=0
ELSE
    IF MOTION_ERROR<>0 THEN
        SET_BIT(0,status_bits) 'Motion error flag
        first_error=ERROR_AXIS
    ENDIF
    alarm_bit=1
ENDIF

'MECHATROLINK axis alarm monitoring
FOR i=0 TO max_axis
    BASE(i)
    VR(servo_status+i*2)=AXISSTATUS
    'if stopped by alarm, notify the alarm code
    IF ATYPE>=40 AND ATYPE<=42 THEN
        IF status_word=2 THEN
            'if no response, notify "communication alarm"
            IF (AXISSTATUS AND 4)<>0 THEN
                VR(servo_alarm+i*2)=$E6
            ELSEIF NOT DRIVE_ALARM(servo_alarm+i*2) THEN
                VR(servo_alarm+i*2)=$E6
            ELSEIF VR(servo_alarm+i*2)=0 THEN
                VR(servo_alarm+i*2)=$bb
            ENDIF
        'if no alarm, notify RUN=$99 or BaseBlock=$BB
        ELSEIF(DRIVE_STATUS AND 8) THEN
            VR(servo_alarm+i*2)=$99
        ELSE
            VR(servo_alarm+i*2)=$bb
        ENDIF
    ENDIF
NEXT i

    sys_error=SYSTEM_ERROR
RETURN

'-------------------------------------------------------
----------------------
```

```
stop_all:

'In this example, if the application program is stopped
suddenly all the
'movements are cancelled and all the axes are set to
BaseBlock. Modify this
'section if you require a different STOP procedure

STOP "APPLICATION"
WDOG=0

FOR i= 0 TO max_axis
    BASE(i)
    IF MARK=0 THEN REGIST(-1)
    AXIS_ENABLE=0
    SERVO=0
    CANCEL(1) 'Cancel NTYPE
    WA(1)
    CANCEL(1) 'Cancel possible program buffer
NEXT i

RAPIDSTOP 'Cancel MTYPE
RETURN

'-----------------------------------------------------
----------------------

start_application:

'Add all the application programs that should be started
with the START signal

RUN "APPLICATION"

RETURN

'-----------------------------------------------------
----------------------

reset_all:

'Uncorrect system setting
IF READ_BIT(15,diag01)=0 THEN GOSUB system_detection

'MECHATROLINK axes reset sequence
FOR i=0 TO max_axis
    BASE(i)
    IF ATYPE>=40 AND ATYPE<=42 THEN
          'Reset sequence for MECHATROLINK communication
error
        IF (AXISSTATUS AND 4)<>0 THEN
            PRINT "Resetting ML alarm"
            GOSUB system_detection
        ENDIF
        'Reset sequence for DRIVE errors
```

```
        IF (AXISSTATUS AND 8)<>0 THEN
                              IF  VR(servo_alarm+i*2)=$81  OR
VR(servo_alarm+i*2)=$CC THEN
                GOSUB absencoder
             ELSE
            'Pending to handle diferently those alarms that
cannot
             'be resetted with DRIVE_CLEAR
             DRIVE_CLEAR
             ENDIF
        ENDIF
    ENDIF
NEXT i

'Reset sequence for AXIS error
DATUM(0)
CLEAR_BIT(0,status_bits)

'MECHATROLINK slaves reset sequence
IF (SYSTEM_ERROR AND $40000)<>0 THEN

'Omron Auto Generated - ML IO
'Warning: Automated code section - any manual code changes
will be lost.

'Omron Auto Generated - ML IO End

    'Same with the other IO devices
ELSEIF SYSTEM_ERROR<>0 THEN
    'Other system error needs initialisation of the system
    EX
ENDIF


RETURN

'-----------------------------------------------------------
-----------------------

warning_seq:

IF READ_BIT(15,diag01) THEN
    'Clear servodrive warning if any
    IF res_bit=1 THEN
        FOR i=0 TO max_axis
            BASE(i)
            IF ATYPE>=40 AND ATYPE<=42 THEN
                IF (DRIVE_STATUS AND 2)>0 THEN DRIVE_CLEAR
            ENDIF
        NEXT i
    ENDIF
ENDIF

RETURN
```

```
'-------------------------------------------------------
----------------------

monitoring:

'Add monitoring depending on the application
RETURN

'-------------------------------------------------------
----------------------

absencoder:
'To be implemented in the future
RETURN

'-------------------------------------------------------
----------------------

system_detection:

status_word=0
action=0
VR(status_bits)=0

'Omron Auto Generated - Units
'Warning: Automated code section - any manual code changes
will be lost.

'Unit Variables reset
VR(diag01)=0
VR(diag02)=0
VR(diag03)=0
VR(unit_detection)=0

VR(unit_detection)=0

'Detection OK
IF  VR(diag01)=0  AND  VR(diag02)=0  AND  VR(diag03)=0  THEN
SET_BIT(15,diag01)

'Omron Auto Generated - Units End

'Start Standard Section

'Drive Parameters

' Axis Parameters

' Variables

' TABLE DATA

'Stop Standard Section

RETURN
```

5-2-2 Initialization program

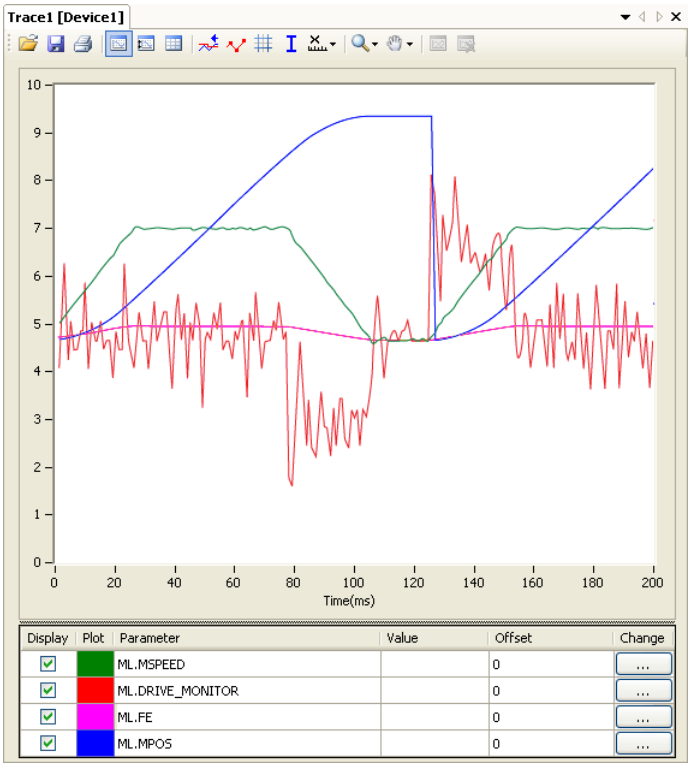
The Initialization program sets the parameters for the axes. These parameters are dependant upon the Motor Encoder resolution and the motor maximum speed.

Note Refer to the Servo Driver and the motor data sheet for this information.

```
'================================================
'EXAMPLE OF INITIALIZATION PROGRAM
'THIS VERSION IS DESIGNED FOR MECHATROLINK-II SERVOS
'ADAPT THIS PROGRAM ACCORDING TO YOUR APPLICATION
'================================================
BASE(x)
restart=0
inertia_ratio=set_load_inertia_ratio

'----------------------------
'EXAMPLE 1
'SGMAH-01AAA61D-OY motor data
'----------------------------
enc_resolution=2^13 '13 bit encoder
max_speed=5000 '5000 rpm max. speed

'----------------------------
'EXAMPLE 2
'SGMAH-01A1A61D-OY motor data
'----------------------------
enc_resolution=2^16 '16 bit encoder
max_speed=5000 '5000 rpm max. speed

'----------------------------
'WRITE PARAMETERS IN THE SERVO
'----------------------------
DRIVE_WRITE($103,2,inertia_ratio) 'Write inertia ratio
DRIVE_READ($110,2,10)
IF VR(10)<>$0012 THEN
    DRIVE_WRITE($110,2,$0012,1)
    'Pn110=0012h (autotuning disabled)
    restart=1
ENDIF
DRIVE_READ($202,2,10)
IF VR(10)<>1 THEN
    DRIVE_WRITE($202,2,1,1)
     'Pn202=1 (gear ratio numerator in the drive. Default
is 4)
    restart=1
ENDIF
DRIVE_READ($511,2,10)
IF VR(10)<>$6548 THEN
    DRIVE_WRITE($511,2,$6548,1)
    'Pn511 set the registration inputs in the Servo Driver
    restart=1
ENDIF
DRIVE_READ($81E,2,10)
IF VR(10)<>$4321 THEN
    DRIVE_WRITE($81E,2,$4321,1)
```

```
    'Pn81E=$4321 To make the Digital inputs in the Servo
Driver
    'available for reading through DRIVE_INPUTS word
    restart=1
ENDIF
IF restart=1 THEN DRIVE_RESET

'----------------------------
'Initial gains For MECHATROLINK_SPEED
'----------------------------
'By experience this setting is a good starting point
P_GAIN=INT(214748.3648*max_speed/enc_resolution)
'This is the optimum value. Set if needed
VFF_GAIN=INT(60000*1073741824/enc_resolution/max_speed)

'----------------------------
'Initial gains For MECHATROLINK_POSITION mode
'----------------------------
'Change the rigidity (Fn001) according to the 'mechanical
system
'Change feedforward gain Pn109 if required

'----------------------------
'Initial parameter of the AXIS
'----------------------------
'If set to 1 (and Pn202=Pn203=1) the UNITS are 'encoder
counts
UNITS=1
'Theoretical FE we will have running the motor at
"max_speed"
'without VFF_GAIN in MECHATROLINK SPEED
FE_LIMIT=1073741824/P_GAIN/UNITS
'SPEED is set to 1/3 of "max_speed
SPEED=(max_speed73)*enc_resolution/60/UNITS
'ACCEL in 200ms from 0 to "max_speed"
ACCEL=SPEED/0.2
'DECEL in 200ms from "max_speed" to 0
DECEL=SPEED/0.2
```

5-2-3 Single axis program



This program is a simple program to run one axis only.

5-2-3-1 Example

```
'GOSUB homing
BASE(0)
DEFPOS(0)
WA(100)
loop:
    MOVE(1440)
    WAIT IDLE
    WA(100)
GOTO loop
```

The units are degrees in this example, therefore:

- 13-bit encoder
- Pn202=32
- Pn203=45
- **UNITS=32**

The graph in the figure is typical for this point-to-point movement with linear acceleration). Note the following:

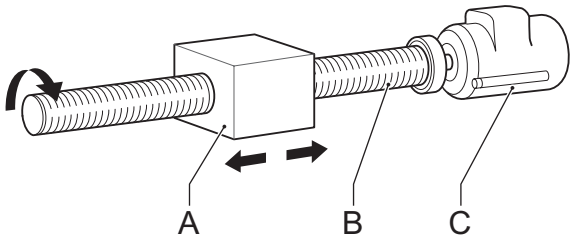
- During linear acceleration, the graph of the position is parabolic (because the speed is a derivative of the position).
- During constant speed, the graph of the position is straight.
- During linear deceleration, the graph of the position is counter-parabolic.
- During stop, the graph of the position is constant.
- When an overflow occurs (**MPOS>=REP_DIST**), the position jumps to 0 if **REP_OPTION=1** or to **-REP_DIST** if **REP_OPTION=0**.

- The Following Error is proportional to the speed if you use only Proportional Gain in the position loop.
- The torque, which is given by **DRIVE_MONITOR** as a percentage of the nominal torque of the motor when you set **DRIVE_CONTROL=11**) is proportional to the acceleration according to the formula:

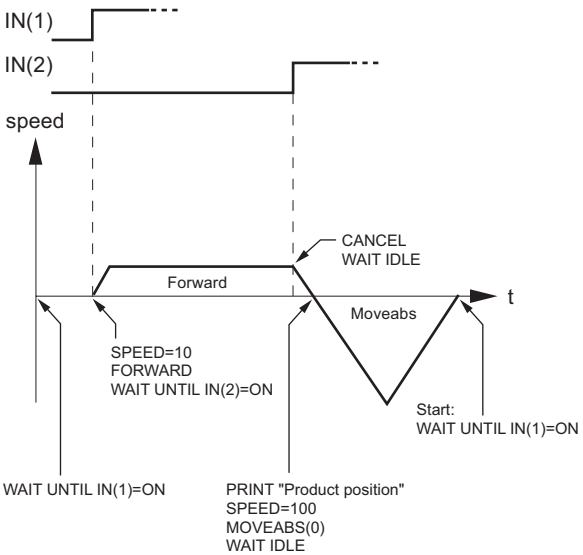
  $Torquetotal = Jtotal \times a + Torquefriction$

  where $Torque_{friction}$ is usually small, α is the angular acceleration, and J the inertia of the system.

5-2-4 Position with product detection



A ballscrew moves forward at a creep speed until it reaches a product, a microswitch (IN(2)) turns on.
The ballscrew is stopped immediately, the position at which the product is sensed is indicated and the ballscrew returns at a rapid speed back to the start position.

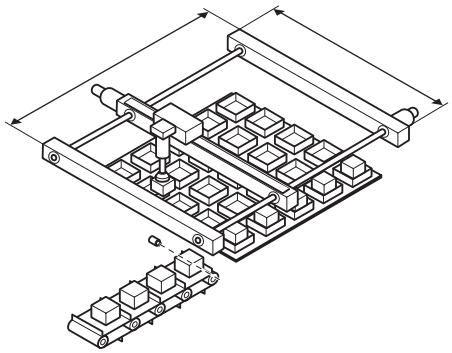


5-2-4-1 Example

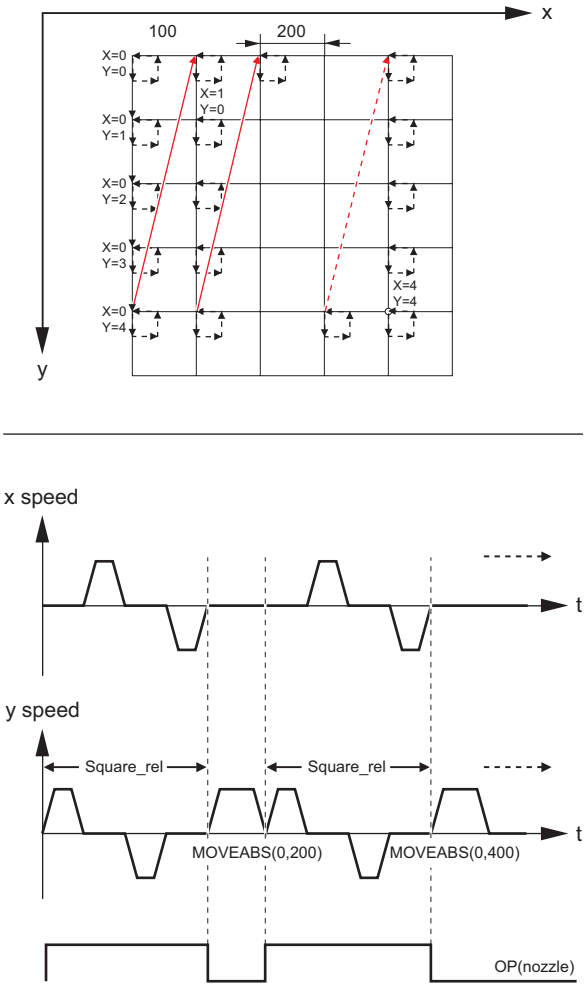
```
start:
    WAIT UNTIL IN(1)=ON
```

```
    SPEED=10
    FORWARD
    WAIT UNTIL IN(2)=ON
    prod_pos=MPOS
    CANCEL
    WAIT IDLE
    PRINT "Product Position : "; prod_pos
    SPEED=100
    MOVEABS(0)
    WAIT IDLE
GOTO start
```

5-2-5 Position on a grid



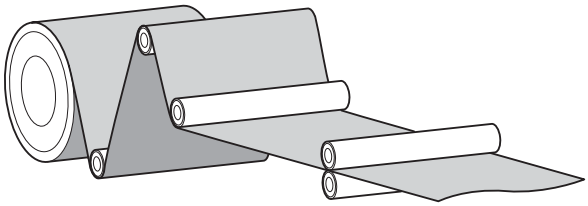
A square palette has sides 1m long. It is divided into a 5 x 5 grid, and each of the positions on the grid contains a box which must be filled using the same square pattern of 100mm by 100mm. A dispensing nozzle controlled by digital output 8 must be turned on when filling the box and off at all other times.



5-2-5-1 Example

```
nozzle = 8
start:
    FOR x = 0 TO 4
        FOR y = 0 TO 4
            MOVEABS(x*200, y*200)
            WAIT IDLE
            OP(nozzle, ON)
            GOSUB square_rel
            OP(nozzle, OFF)
        NEXT y
    NEXT x
GOTO start
square_rel:
    MOVE(0, 100)
    MOVE(100, 0)
    MOVE(0, -100)
    MOVE(-100,0)
    WAIT IDLE
    WA(1000)
RETURN
```

5-2-6 Bag feeder program

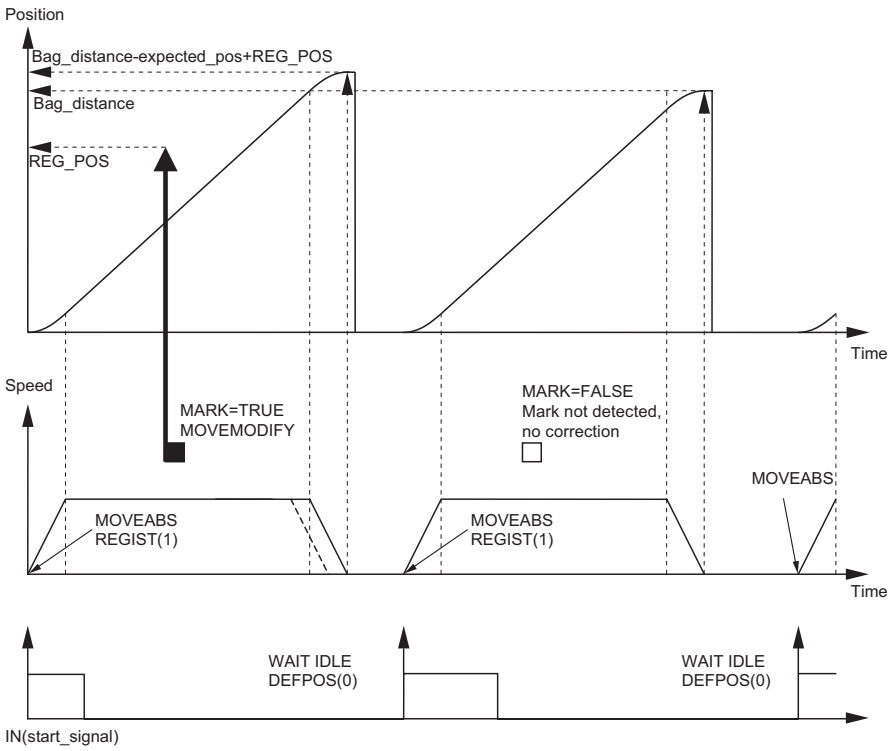


A bag feeder machine feeds plastic film a fixed distance that is set by the operator. The figure shows a typical bag feeder that is part of the machine.

Bag feeder machines have two modes.
- Without mark: Forward feeds the film a set distance, for films of a flat colour
- With mark: Forward feeds the film to a printed mark on the film.

The program in this section shows the typical code for a bag feeder machine.

5-2-6-1 Example




```
'=============================================
'BAG FEEDER program
```

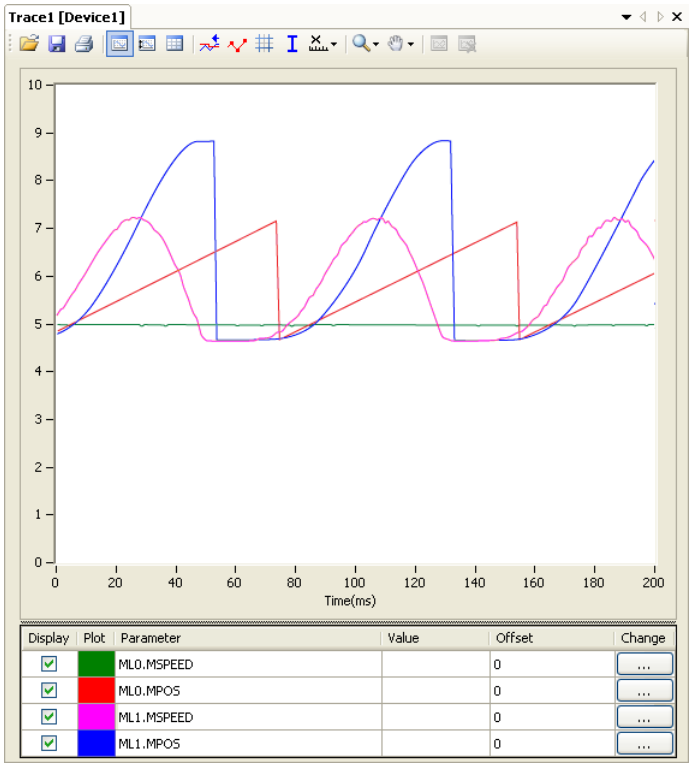
```
'================================================
'Working with marks, if any mark is missing, feed the
'theoretical distance. But if the mark is missing for
'a number of consecutive bags, stop the operation.
'A digital output is activated a certain time to cut
'the bag.
'================================================

'Variable initialisation
start_signal=7
max_fail=3
program_alarm=0
failed=0
feeder_axis=2
BASE(feeder_axis)
'Position counter (MPOS,DPOS) goes from 0 to 999999
'and 0 again
UNITS=27
SPEED=100
ACCEL=1000
DECEL=1000
REP_DIST=1000000
REP_OPTION=1
SERVO=ON
WDOG=ON

'Main program
loop:
    'Define current position as zero
    DEFPOS(0)

    'Wait for rising edge in Digital Input
    '"start_signal"
    WAIT UNTIL IN(start_signal)=0
    WAIT UNTIL IN(start_signal)=1

    'Move bag length
    MOVEABS(bag_distance)
     WAIT UNTIL MTYPE=2 'To verify that the MOVEABS is
being executed

    'If we work with Mark, activate the trigger
    'MARK=FALSE when triggered and TRUE when not triggered
    IF work_with_mark AND MARK THEN
        REGIST(1)
        WAIT UNTIL MARK=0
    ENDIF

    'Wait until movement finished or mark detected
    WAIT UNTIL MTYPE=0 OR (MARK AND work_with_mark)

    'Working with mark
    IF work_with_mark THEN
         IF MARK THEN 'If the mark has been detected, the
position is corrected
            MOVEMODIFY(bag_distance-expected_pos+REG_POS)
```

```
                failed=0

            ELSE 'If the mark has not been detected
                PRINT "Mark not detected"
                failed=failed+1
                      IF failed>max_fail THEN 'After several
consecutive misdetection stop the application
                    PRINT "Mark definitelly lost"
                    program_alarm=3
                    STOP
                ENDIF
            ENDIF
        ENDIF

        'Wait until the feed movement has finished
        WAIT IDLE
GOTO loop
```

5-2-7 CAM table inside a program

It shows how to create a CAM table inside a program, and use the **CAMBOX** motion command.
The profile used is the COS square one. This is a quite typical profile for feeder-type applications as:

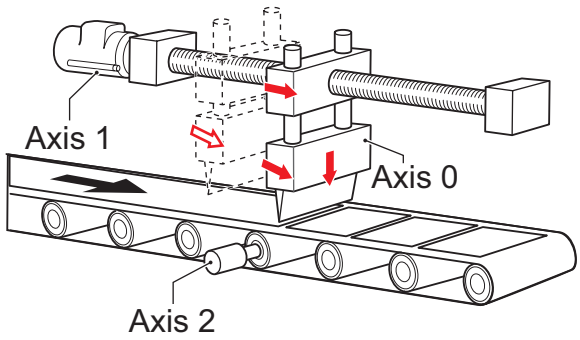
- The motion provides a smooth acceleration without sudden acceleration changes, so the material slip is minimized
- It gives a fast deceleration so the cycle time is reduced. During deceleration there is no material slip and the friction helps to the stop to zero.



5-2-7-1 Example

```
start:
    GOSUB filltable
    WDOG=1 'Set servos to RUN
    BASE(1)
    SERVO=1 'Enable position loop in axis 1
    BASE(0)
    SERVO=1 'Enable position loop in axis 0
    'The position counter counts from 0 to 11999
    'and then back to 0 again
    REP_OPTION=1
    REP_DIST=12000
    SPEED=200
    FORWARD

BASE(1)
loop:
    CAMBOX(in_tbl,end_tbl,1,lnk_dst,master,opt,start)
    WAIT IDLE
GOTO loop

filltable:
    'The shape of the CAM is stored in TABLE(0) to
    'TABLE(360)
    npoints=360
    in_tbl=0
    end_tbl=in_tbl+npoints
    'Distance of the master to make the CAM
    lnk_dst=10000
    'Master axis
    master=0
    'The CAM start exactly when the master reaches
    'position "start"
    opt=2
    start=1000

    k=100
    'Fill the TABLE with the suitable waveform
    FOR i= in_tbl TO end_tbl
        TABLE(i,(k*(COS(PI*i/npoints)-1))^2)
    NEXT i
RETURN
```

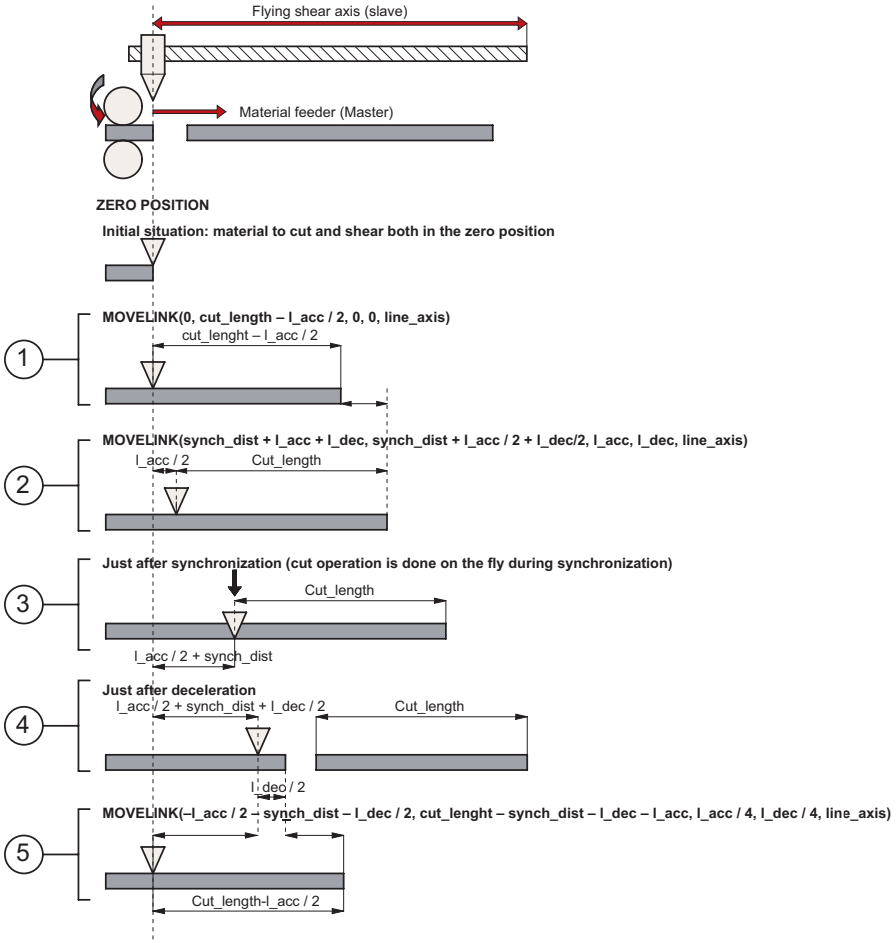
5-2-8 Flying shear program



An example of the Flying shear program. In this application there are three axes:

- Axis 0, shear_axis, the advancement of the shear.
- Axis 1, flying_axis, is the flying shear.
- Axis 2, line_axis, transports the material.

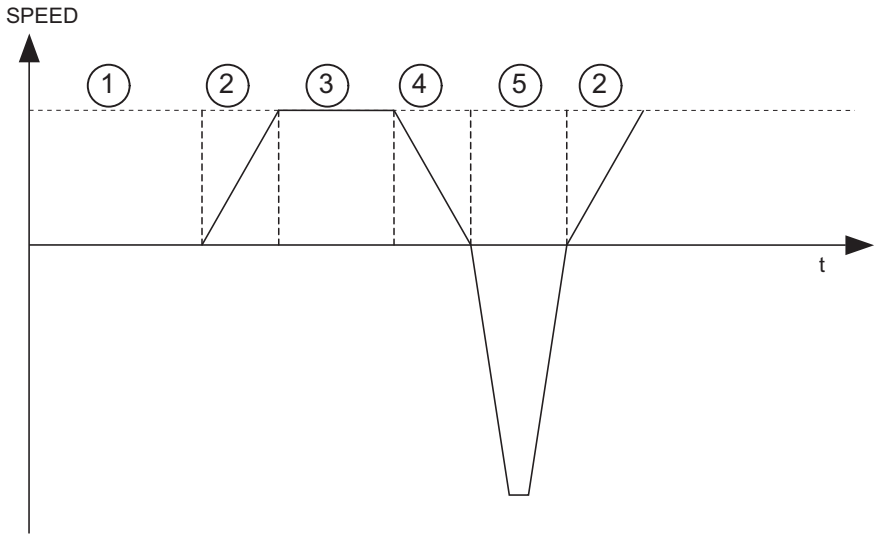
5-2-8-1 Example


Flying shear axis (slave)
Material feeder (Master)
ZERO POSITION
Initial situation: material to cut and shear both in the zero position
MOVELINK(0, cut_length – l_acc / 2, 0, 0, line_axis)
cut_lenght – l_acc / 2
1
MOVELINK(synch_dist + l_acc + l_dec, synch_dist + l_acc / 2 + l_dec/2, l_acc, l_dec, line_axis)
l_acc / 2
Cut_length
2
Just after synchronization (cut operation is done on the fly during synchronization)
Cut_length
l_acc / 2 + synch_dist
3
Just after deceleration
l_acc / 2 + synch_dist + l_dec / 2
Cut_length
l_dec / 2
4
MOVELINK(–l_acc / 2 – synch_dist – l_dec / 2, cut_lenght – synch_dist – l_dec – l_acc, l_acc / 4, l_dec / 4, line_axis)
Cut_length-l_acc / 2
5

```
'================================================
```

```
'FLYING SHEAR program
'================================================
'Typical example of a flying shear application.
'One axis (line_axis) transport the material
'Second axis (flying_axis) is the flying shear itself
'Third axis (shear_axis) is the shear advancement
'The distance in synchronization must be long enough
'to allow the cut at maximum speed.
'The return of the flying shear is done at such a
'speed that the wait time is zero (optimization of
'the movement).
'Again it is assumed that everithing has been
'calculated to not exceed the maximum motor speed at
'maximum line speed
'================================================
cut_counter=0
line_axis=2
shear_axis=0
flying_axis=1

SERVO AXIS(line_axis)=ON
SERVO AXIS(flying_axis)=ON
SERVO AXIS(shear_axis)=ON
WDOG=ON

'FIRST CYCLE

'Make a first material cut
MOVEABS(end_pos) AXIS(shear_axis)
WAIT UNTIL MTYPE AXIS(shear_axis)=2
WAIT IDLE AXIS(shear_axis)

'First time we have a certain wait time because the
'material has been just been cut
wait_distance=cut_lenght-l_acc/2
MOVELINK(0,wait_distance,0,0,line_axis) AXIS(flying_axis)
WAIT UNTIL MTYPE AXIS(flying_axis)=22

'We start the line
FORWARD AXIS(line_axis)

loop:

    'Update the line speed every cycle
    SPEED AXIS(line_axis)=line_speed

    'Cutting movement at synchronized speed
    line_cut=synch_dist+l_acc+l_dec
    shear_cut=synch_dist+l_acc/2+l_dec/2
       MOVELINK(shear_cut,line_cut,l_acc,l_dec,line_axis)
AXIS(flying_axis)
    WAIT UNTIL MPOS AXIS(flying_axis)>l_acc/2

    'Activate the shear when it is in synchronization with
the line
    'Slow speed to cut
```

```
    SPEED AXIS(shear_axis)=cut_speed
    MOVEABS(end_pos) AXIS(shear_axis)
    MOVEABS(0) AXIS(shear_axis)
    WAIT UNTIL NTYPE AXIS(shear_axis)=2
    'Fast speed to return
    WAIT LOADED AXIS(shear_axis)
    SPEED AXIS(shear_axis)=return_speed

    cut_counter=cut_counter+1inch

     'Return back synchronized with the master in such a
way
    'that there is no wait time
    line_back=cut_length-synch_dist-l_dec-l_acc
    shear_cut=l_acc/2+synch_dist+l_dec/2)
              MOVELINK(-shear_cut,line_back,l_acc/4,l_dec/
4,line_axis) AXIS(flying_axis)

GOTO loop
```



The speed-time graph shows the steps of the above example. The steps are:

1 The initial cycle: the slave waits for the right length in the product to cut (cut_length – distance_to_accelerate / 2). It is necessary to divide distance_to_accelerate when we use the **MOVELINK** command, because when we synchronize, the master moves twice the distance of the slave.
2 The slave accelerates to synchronize with the master. When the acceleration finishes, the relative distance between the edge of the product and the shear is cut_length.
3 This is the synchronization part: the relative distance between the edge of the product and the shear remains the same. The cut in the material is made. This gives a new material edge.
4 The deceleration part: the material continues, and the shear stops.
5 Move back at high speed: the distances are calculated such that when the slave reaches it original position, the edge of the product is in the correct position to start a new cut.A

A new movement starts (step 2).

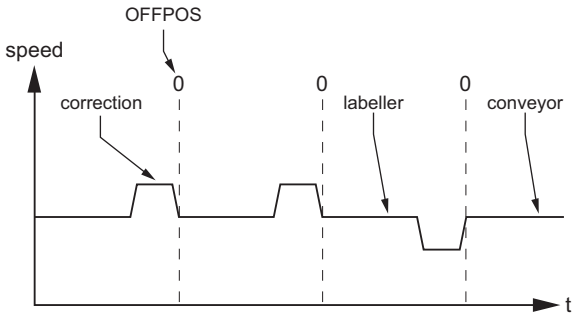
5-2-9 Correction program

This application is for a rotary labeller. The constants are:

- The product arrives on a conveyor (master axis) that runs at a constant speed.
- A rotary labeller that is synchronized 1:1 to the conveyor, attaches the labels.
- The distance between products is fixed and mechanically guaranteed.

The distance between labels is never exactly constant so, a correction is needed. This is done by superimposing a virtual axis onto the movement of the labeller.
The difference between the expected position and the actual position is measured with a photocell. This is the correction factor.
Every time a correction is made, the origin position is updated accordingly.

5-2-9-1 Example




```
conveyor=0
labeller=1
virtual=15
SERVO AXIS(conveyor)=1
SERVO AXIS(labeller)=1
WDOG=1

BASE(labeller)
CONNECT(1,conveyor)
ADDAX(virtual)
FORWARD AXIS(conveyor)
REGIST(1)
WAIT UNTIL MARK=0

loop:
    WAIT UNTIL MARK
    correction=REG_POS+expected_pos
    MOVE(correction) AXIS(virtual)
    WAIT IDLE AXIS(virtual)
    OFFPOS=-label_length+correction
    REGIST(1)
    WAIT UNTIL MARK=0
GOTO loop
```

SECTION 6
Troubleshooting

6-1 System errors

Indication	Problem	Solution
ERC LED flashes at startup	Hardware error	Replace the CJ1W-MCH72
ERC LED flashes during execution (RUN LED is on)	Low or empty battery	Replace the battery
ERC LED is on	Error log access error	Replace the CJ1W-MCH72
ERH LED is on	Communication error between PLC CPU and CJ1W-MCH72	Turn the PLC system off and then on

6-2 Data exchange errors

If the PLC CPU memory area and the CJ1W-MCH72 memory area are configured for data exchange, but it appears that no data is exchanged, try one of the following solutions:

- Check the range of the memory areas. Make sure that the start address and the end address of the memory area are in range.
- Check the exchange direction of the memory area:
  - Make sure that data for PLC output units is transferred from the CJ1W-MCH72 to the PLC CPU.
  - Make sure that data from PLC input units is transferred from the PLC CPU to the CJ1W-MCH72.

A

Absolute
EnDat, 27
SSI, 27
Absolute encoder
Wiring, 44
Architecture, 15
Axis sequence, 23
Axis type, 23

B

Bag feeder program example, 307
BASIC commands, 65
BASIC program, 3
Battery, 35
Replace, 36
Buffer types, 29
Buffers, 16, 29

C

CAM table example, 309
Command
Axis, 65
Communication, 69
I/O, 69
Program, 71
Program control, 72
System, 72
Task, 75
Communication, 16
Complex profile, 24
Components
CJ1W-MCH72, 33
Configuration examples, 19
Connector
Encoder, 37
I/O, 37
MECHATROLINK-II, 37
Constants, 69
Correction example, 314
CPU task, 3
Cycle time, 2, 16

D

Data exchange, 55
Configurable data, 57
Configuration, 56
Control data, 56
Errors, 315
Memory areas, 55
Status data, 56
Definition
CPU task, 3
Cycle time, 2
Motion sequence, 2
Process, 3
Program, 3
Servo period, 2
Description Motion buffers, 16

E

Encoder connector, 37
Wiring, 41
Encoder output, 27
EnDat, 27
Errors
Data exchange, 315
System, 315
Example
Bag feeder program, 307
CAM table, 309
Configuration, 19
Correction program, 314
Flying shear program, 311
Gain settings, 245
Homing, 274
Initialization program, 301
Motion buffers, 30
Multi-tasking, 21
Origin search, 274
Position mode, 255
Position on a grid, 305
Position with product detection, 304
Registration, 279
Servo Driver characteristics, 270
Servo period, 17
Setting units, 260
Shell program, 293
Single axis program, 303
Speed mode, 246
Startup program, 243
Tracing and monitoring, 287
Explanation
Communication, 16
Cycle time, 16
Motion buffers, 29
Motion sequence, 15, 22
Multi-tasking, 21
Peripherals, 16

Program control, 15
Servo period, 17

F

FINS, 58
Error Data Read, 64
Parameter Area Read, 61
Parameter Area Write, 62
Read, 59
Run, 63
Stop, 63
Write, 60

Flying shear example, 311

Function
I/O, 69
Mathematical, 70
System, 72

G

Gain example, 245

H

Homing example, 274

I

I/O connector, 37
Wiring, 38

Incremental encoder
Hardware PSWITCH, 43
Input, 42
Output, 44
Registration, 43
Wiring, 42

Inertia ratio, 30

Initialization example, 301

Installation, 48

M

MECHATROLINK-II
Connecting slaves, 51
Inverter as axis, 27
Position control, 25
Specifications, 54
Speed control, 25
Torque control, 26

MECHATROLINK-II connector, 37

Modifier
Slot, 72

Motion buffers, 16, 29

Motion control, 3
Continuous path, 7
Electronic gearing, 8
Point-to-point, 4

Motion sequence, 2, 15

MTYPE, 29

Multi-tasking example, 21

N

NTYPE, 29

O

Operand, 70
Mathematical, 70

Origin search example, 274

P

Parameter
Axis, 66
Communication, 69
I/O, 69
Slot, 72
System, 74
Task, 75

Peripherals, 16

Position control, 25

Position loop algorithm, 23

Position mode example, 255

Position on a grid example, 305

Position reference, 26–27

Position with product detection example, 304

Priority
Program control, 20

Process, 3

Process 0, 20

Process buffer, 29

Profile generator, 23

Program control, 15

Program control priority, 20

R

Registration example, 279
Resonant frequency, 30
Rigidity, 30

S

Servo axis, 26
Servo Driver characteristics example, 270
Servo period, 2
  Examples, 17
  Rules, 18
Servo system, 12
  CJ1W-MCH72 operation, 13
  Motion control algorithm, 13
  Semi-closed loop, 12
Shell example, 293
Single axis example, 303
Specifications
  Dimensions, 52
  Encoder connector, 41
  Encoder interface, 54
  I/O connector, 39
  MECHATROLINK-II, 54
  System, 53
  Unit, 52
Speed control, 25
Speed mode example, 246
Speed reference, 26–27
  EnDat, 27
  SSI, 27
SSI, 27
Startup example, 243
Status LEDs, 33, 35, 50
Stepper output, 26
System architecture, 15

T

Torque control, 26
Tracing and monitoring example, 287
Troubleshooting
  Data exchange errors, 315
  System errors, 315

U

Unit components, 33
Unit number, 35
  Create I/O table, 50
  Setting, 49
Units example, 260

V

Virtual axis, 24

W

Wiring, 38
Word allocations, 35

Revision history

A manual revision code appears as a suffix to the catalog number on the front cover of the manual.

Cat. No. I55E-EN-01

Revision code

The following table outlines the changes made to the manual during each revision. The page numbers of a revision refer to the previous version.

Revision code	Date	Revised content
01	August 2008	First version

OMRON

Authorized Distributor:

Cat. No. I55E-EN-01

Note: Specifications subject to change without notice.

Printed in Europe